UNIVERSITY OF CALGARY

Index-Calculus Algorithms for Computing Class Groups of Quartic Number Fields

by

David Marquis

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES

IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE

DEGREE OF DOCTOR OF PHILOSOPHY

GRADUATE PROGRAM IN COMPUTER SCIENCE

CALGARY, ALBERTA

APRIL, 2024

Abstract

We address the problem of quickly computing the class group and unit group for quartic number fields of large discriminant. Index-calculus algorithms are the fastest way to solve this problem for arbitrary number fields. Our focus is primarily on improvements to relation generation, one of the main stages in an index-calculus algorithm. In the quadratic case, the self-initialization approach to relation generation developed by Jacobson has been very successful, but applying this idea to quartic fields has not been attempted. We present a novel generalization of this approach that is applicable to quartic number fields. Additionally, we characterize the efficiency of our method in terms of the size of the roots of the field's defining polynomial. We discuss our implementation of a complete index-calculus algorithm using this approach.

Our implementation's relation generation produces relations significantly faster than the current state-of-the-art, `Magma`. Our implementation of the complete algorithm, including the improved relation generation, is faster than `Magma` for number fields whose defining polynomial has small roots, and is comparable for typical number fields.

Preface

This thesis is an original work by the author. No part of this thesis has been previously published.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Contents

1.1 Introduction

A number field $\mathbf{K}$ is an extension of the rational numbers that can be represented as $\mathbf{Q}[X]/(f)$ where f is an irreducible polynomial over the rationals. The degree d of $\mathbf{K}$ is defined as the degree of f. The **class group** of $\mathbf{K}$ is the quotient of the group of fractional ideals by the subgroup of principal fractional ideals. The elements of the class group are equivalence classes of ideals, where two ideals are equivalent if their quotient is a principal ideal. The **unit group** of $\mathbf{K}$ consists of all the elements in the ring of integers of $\mathbf{K}$ with a multiplicative inverse that is also in the ring of integers of $\mathbf{K}$.

In number fields there is a significant difference between the ideals generated by algebraic numbers and the algebraic numbers themselves. The class group and unit group of $\mathbf{K}$ help to understand this difference. As Neukirch [Neu99, Chapter 2] says,

> "the class group measures the expansion that takes place when we pass from numbers to ideals, whereas the unit group measures the contraction in the same process."

These two groups are so fundamental that Cohen [Coh93, Chapter 4] puts computing them amongst the five main computational problems of algebraic number theory. This thesis addresses the computational challenges of determining the class group and unit group, focusing on developing an efficient algorithm for computing them.

Algorithms using the index-calculus framework are the fastest for solving this problem and one of the most important stages in this framework is relation generation. Although there are index-calculus algorithms applicable to arbitrary degrees, researchers have been able to develop much faster relation generation for quadratic fields by taking advantage of the specialized degree. This thesis gives a novel approach to relation generation that generalizes the self-initialization approach of Jacobson [Jac99b] for quadratic fields to the case of arbitrary degree number fields. To demonstrate the effectiveness of our relation generation approach, we chose to focus on the case of quartic number fields (cubic fields being under investigation by Luo [Luo24] concurrently with our research). Our primary performance metric is the time per relation (defined in Section 1.3), which indicates the speed at which useful relations are generated. In addition, a new implementation of the entire algorithm, using our relation generation, allows us to compare against existing systems.

Motivation for the problem of computing class groups and unit groups of low degree number fields is discussed in Section 1.2. The contributions of this thesis are discussed in Section 1.3. In Section 1.4 we describe the structure of this thesis.

1.2 Motivation

We briefly outline some of the many applications of the class group and unit group of number fields of small degree.

Computing class groups has been associated with the security of several cryptographic algorithms. Beginning with Buchmann and Williams [BW90], a number of cryptosystems (see [JW09, Chapter 13]) were developed that use the difficulty of computing class groups or unit groups of quadratic fields as the basis for public-key encryption and digital signature algorithms. The security of many of these schemes relies on the presumed difficulty of computing class groups and unit groups. Like other computationally hard problems that cryptosystems rely on for their security, the study of the algorithms for solving these problem is essential. It forms the primary basis for our confidence in the security of such cryptosystems.

More recently, isogeny-based cryptography has become a promising direction in post-quantum cryptography. Computing class groups has become a component in cryptography based on isogenies of elliptic curves (see [CLM$^+$18] for background on this approach). Using this connection, some cryptographic protocols have been created for which knowledge of the class group enables faster execution than similar protocols not utilizing the class group. For example, in [BKV19] a digital signature algorithm is given which requires knowledge of the class group of an imaginary quadratic field. The resulting digital signature protocol is hundreds of times faster and the signature is three times smaller than other digital signature algorithms that are similar but do not require knowledge of the class group.

For quartic fields it is possible that knowledge of the class group can also be used as the basis of cryptographic protocols. Isogeny-based cryptography for genus two hyperelliptic has been considered in the literature (see [CS20]) so cryptography involving hyperelliptic curves may make use of knowledge of the class group of a quartic field in the future. A fast class group algorithm for degree four number fields could be useful to make such protocols efficient.

The study of the class group and unit group is also interesting in its own right. Many basic questions have not been resolved, such the famous conjecture of Gauss that there are infinitely many quadratic number fields of positive discriminant with class number one. The tabulation of class groups allows the validation of important conjectures such as those proposed by Cohen-Martinet [CM87]. These conjectures extend the well-established Cohen-Lenstra heuristics, originally formulated for quadratic number fields, to fields of higher degrees. An example is [MJ16] which tabulates all class groups of imaginary quadratic fields with absolute value of discriminant up to 2^{40}. Databases of such tabulations are available online, one of them being the `LMFDB` [LMF24b] project.

Lastly, knowledge of class groups and unit groups is often a necessary step for computing algebraic objects. Such objects include solutions to Diophantine equations. For example, solving the Pell equation amounts to computing the unit group of a quadratic number field [JW09]. As another example, one method for solving Thue-Mahler equations makes use of knowledge of the class group of an associated number field. Another type of algebraic object that involves class groups are the trace formulas for certain modular forms [BC18]. Thus fast algorithms for computing class groups can also be used for computing these formulas.

1.3 Contributions

The absolute value of the discriminant of a number field is a measure of the number field's size. The calculation of the class group and the unit group becomes difficult as $|\Delta|$ grows, with the most efficient algorithms following an index calculus framework. One of the most time-consuming stages of an index-calculus algorithm is **relation generation**, which is the primary focus of this thesis. Before continuing, we emphasize that our goal in this thesis is to consider the efficiency of algorithms in practice.

To explain relation generation we need to introduce a few concepts. Rational numbers are called **B-smooth** if they factor over a set of rational primes called a **factor base** that are smaller than B. We call an ideal B-smooth if it factors over a factor base of prime ideals of norm less than B and the set of primes below the factor base primes are called the rational factor base. **Relation generation** consists of finding algebraic numbers α so that the principal ideal (α) is B-smooth. If an ideal generated by a relation generation algorithm has B-smooth norm it is almost always B-smooth in practice. This suggests that a good way to find smooth ideals is to reduce the problem to finding smooth norms.

The existing research on relation generation falls into two categories. The first is the development of algorithms and implementations applicable to arbitrary degree number fields. The second is algorithms for quadratic number fields. As mentioned, these algorithms use techniques specific to degree 2 and they demonstrate that it is possible to achieve significant improvements by specializing to a fixed, small degree. These algorithms are designed based on standard heuristic assumptions that we will discuss in the next chapter. The approach to relation generation developed by Jacobson in [Jac99b] is the self-initializing quadratic sieve (SIQS). SIQS has been very successful but similar techniques have yet to be employed for fields of higher degree. This is the gap this thesis intends to fill.

Following Jacobson's sieving approach we fix an integer A that is B-smooth and an algebraic integer ω_1. We fix an integer called the **sieving radius** E and test the ideals $(xA + y\omega_1)$ for integers x and y in the interval $[-E, E]$ for smoothness. We first check if $(xA + y\omega_1)/A$ is B-smooth while also obtaining the factorization of its norm using sieving. If the norm is B-smooth then it is easy to confirm that $(xA + y\omega_1)$ is B-smooth and to factor it into prime ideals.

A good way to compute the norms more quickly is to compute the **norm form** of $(xA + y\omega_1)$ which is the polynomial $g(X, Y)$ so that $g(x, y) = N(xA + y\omega_1)$. Then rather than using the naive approach of evaluating g for every x and y in $[-E, E]$ we can use sieving. Sieving uses the same general idea as the Sieve of Eratosthenes in order to find the B-smooth values of $g(x, y)$ in less time than the naive approach. In its usual form the Sieve of Eratosthenes finds primes in $[-E, E]$ but the means by which it does this is to identify the composite numbers in this interval. Only a slight modification is needed to obtain the factorization of the composites in the interval. The idea is that if we have a prime p that divides $g(x, y)$ then p also divides other values of this polynomial at x', y' that differ by a multiple of p from x and y. We first define a two-dimensional array corresponding to coordinates of x and y. We then iterate through the primes in the rational factor base and mark the array at x, y as being divisible by p. After going through all the primes in the factor base we have found information about the divisibility of each number in the interval by primes in the factor base. With a little more work we obtain the B-smooth part of each number in the interval which allows us to identify the B-smooths.

There are three primary contributions of this thesis: The first contribution, described in Section 1.3.1, is an approach that maximizes the number of B-smooth ideals found, under standard heuristic assumptions, when sieving the norm form of elements in an ideal with a certain type of prime factors. The second contribution, described in Section 1.3.2, extends a sieving technique called self-initialization that employs multiple sieving polynomials from quadratic fields to higher degree number fields. The third contribution, described in Section 1.3.3, is the development of a standalone implementation incorporating modern optimizations and designed for ease of modification and extension. We begin with describing the general setup used by our sieving approach.

1.3.1 Norm Bound

In order to ensure a high probability of smoothness for the numbers $g(x, y) = \mathrm{N}(xA + y\omega_1)/A$ we try to keep $g(x, y)$ as small as possible for a given E through the choice of the size of A. For Jacobson's SIQS a bound of [Sil87] is used. Using this bound gives an upper bound of $O(\sqrt{|\Delta|} \cdot E^2)$ on the norms that are produced. We give an analogue of this which applies to arbitrary degree number fields when the ideal that A, ω_1 are drawn from factors into a product of a certain type of prime ideals.

Authors of previous work on sieving algorithms for number fields of degree larger than two have omitted such a bound. This omission makes the assessment of the performance of these approaches more difficult. We consider that both [Nei02] and [BF12] give timings for two different classes of defining polynomials. The timings between the two classes are very different which raises the question of the cause of this difference. Their omission of a norm bound makes it hard to know how much of this difference is due to norm size.

Furthermore, the performance of their sieving approaches depends on the height of elements that they search for in the maximal order. Finding elements in the maximal order of small height is a generalization of the problem of finding a monic defining polynomial of small height, which is a difficult problem. There is no reason to think their approaches will find elements with height smaller than say $\sqrt{|\Delta|}$ in which case the size of the norms they produce will be larger than $\sqrt{|\Delta|} \cdot E^d$ (recalling that d is the degree of the number field).

The determination of the norm bound depends on the representation of the number field $\mathbf{Q}[X]/(f)$, in other words the choice of f. Every number field has infinitely many defining polynomials and so we benefit from identifying a good one. In quadratic fields the defining polynomial $f = X^2 - t$ for t a squarefree integer is a good one to use for SIQS because its roots are small compared to $|\Delta|$. Similarly, the self-initialization approach that we give also uses an f whose roots are small. To make this precise we introduce the idea of a λ-reduced polynomial where λ is a measure of the size of the roots. We give computational evidence that for most quartic number fields a defining polynomial with $\lambda = \frac{1}{(2\cdot(4-1))} = \frac{1}{6}$ can be found through the use of a polynomial reduction algorithm of [CD91] named `polredabs`. However, defining polynomials with smaller λ exist for some families of number fields. For degree four fields we show that there are families of number fields with λ as small as $\frac{1}{(4\cdot(4-1))} = \frac{1}{12}$. This leads us to define the **general case** as λ approximately equal to $\frac{1}{6}$, while the

special case as λ significantly less than $\frac{1}{6}$.

Our primary contribution regarding the norm bound is Theorem 4.2. As mentioned, it is an analogue of the bound for quadratic fields of [Sil87] which applies to arbitrary degree number fields. It applies to ideals $\mathfrak{a}$ that are products of degree one prime ideals. In this theorem we give an expected upper bound $O(|\Delta|^{(d-1)\lambda}E^d)$ on the norms that we test for smoothness. This bound specializes to $O(\sqrt{|\Delta|}E^4)$ for general case quartic number fields. This is the first time the size of the norm in a sieving approach for number fields of degree larger than two has been bounded.

The bound is important for three reasons:

- This bound is comparable to the bound $O(\sqrt{|\Delta|}E^2)$ in the quadratic case.
- Previous sieving algorithms could potentially give much larger norms than this bound.
- It lets us see the effect of λ on the size of the norms we test for smoothness.

1.3.2 Self-initialization Relation Generation

Our second contribution is to give a version of self-initialization that generalizes the approach of Jacobson in the quadratic case so that it works in any fixed degree. The first step is to give an approach using multiple sieving polynomials that permits our approach to self-initialization to be added on top of it.

The motivating idea for using multiple sieving polynomials rather than one is that it allows a smaller sieve radius E to be used. This in turn decreases the size of the numbers being tested for smoothness.

Jacobson starts by defining an approach to relation generation which he calls the multiple polynomial quadratic sieve (MPQS). Our approach is based on MPQS but there are some important differences. One of these is that Jacobson uses the fact that quadratic fields are a Galois extension of the rational numbers in order to compute the norm form of the sieving elements A, ω_1. For non-Galois fields such a calculation would not work. Our approach is to compute the norm form for an arbitrary pair of sieving elements using a multivariable resultant. The resulting expression has low degree in each indeterminate and a small number of terms. Thus the sieving polynomial for each pair of sieving elements can be computed simply by evaluating this simple expression. This is much faster than the interpolation method used in prior work (as described in the next paragraph).

Our approach also differs greatly from the multiple polynomial approach of [Nei02] and [BF12]. For computing the norm form of a pair of sieving elements these authors use an interpolation method. The interpolation approach must be reused for each pair of sieving elements and so becomes expensive as the number of sieving polynomials grows. Another difference is that our approach provides randomization of the relations by using many initial ideals $\mathfrak{a}$. In the other approaches the majority of relations are generated by sieving a single ideal within which the pair of sieving elements is randomized using LLL lattice reductions.

We now turn to the self-initialization strategy that is added on to the multiple polynomial approach. The problem with MPQS is that changing the sieving polynomials has a cost. The data we need for sieving are the roots of the sieving polynomial modulo the primes in the rational factor base and these are somewhat expensive to compute. **Self-initialization** is a strategy in which we calculate the coefficients and roots of the subsequent sieve polynomial from the previous one through a handful of quick arithmetic operations.

As mentioned, our self-initialization approach generalizes the approach of Jacobson [Jac99b] in the quadratic case so that it works for arbitrary degree. One change that is necessary to accomplish this is to modify the algebra of the Gray code enumeration so that it will work for an arbitrary defining polynomial. SIQS relies on the symmetry of the defining polynomial $X^2 - t$. Thus to generalize SIQS to any fixed degree number field we use different terms for both the initial value of the Gray code and for the terms that are added in each iteration.

Another modification we make is in how we relate the roots of one sieving polynomial to the roots of the next sieving polynomial in the enumeration. Self-initialization approaches in the quadratic case rely on the fact that quadratic fields are Galois extensions to relate the roots of the sieving polynomials. As this approach will not work for non-Galois extensions, we instead relate the roots of the sieving polynomials using an algebraic property of the norm form.

1.3.3 Implementation & Computations

Our implementation [Mar24] is composed of 10,000 lines of our own code. To implement an efficient version of index-calculus we take advantage of the fastest algorithms available today. Our code incorporates optimizations like a large prime variation and sparse linear algebra techniques that speed up the linear algebra stage. The standalone design of our implementation makes it easy to modify and extend.

The decision to develop a new implementation was guided by the desire to provide a basis for straightforward benchmarking without the constraints of existing software libraries. Thus our implementation is independent of other implementations for computing class groups and unit groups.

This implementation is used to validate the two contributions we have discussed above. Apart from timings of our implementation, we also make comparisons of our implementation with `Magma`'s implementation as it is the current state-of-the-art.

Regarding the norm bound we discussed in Section 1.3, in our computations in Section 6.2 we use two sets of defining polynomials for pure number fields that have different values of λ, ranging from the typical case of 1/6 (obtainable for arbitrary quartic fields) to 1/12 for a certain class of pure quartic fields. This allows us to isolate the effect of λ on the algorithm's total time. The timing results illustrate the strong effect of λ on total running time.

Our methodology for timing the algorithm is designed to model arbitrary number fields by applying `polredabs` to polynomials with bounded coefficients. We consider relation generation using the metric of time per relation as a way to measure the quality of a relation generation approach. This is defined in terms of the set of "useful" relations, the ones included in the relations matrix M to make it full rank. The reasons we consider time per relation are: first, that the speed of generating useful relations is crucial to the success of a relation generation approach. Second, this metric is easily computed for `Magma`'s implementation as well as our own. By this metric we find our self-initialization relation generation is faster than `Magma`'s.

Regarding the self-initialization approach we discussed in Section 1.3, it is important to note the distinct nature of our approach to relation generation compared to those in existing index-calculus algorithms for quartic number fields. Our method does not merely augment an existing successful algorithm with self-initialization; it represents a novel approach. As such it is important to validate this approach. Our main hypothesis was that our self-initialization method would have a smaller time per relation than prior work. The validation of this hypothesis is an important step toward the potential future integration of self-initialization into index-calculus algorithms.

We give timings of the time per relation of our self-initialization implementation and compare it to `Magma`'s time per relation. Our results show that our time per relation is often more than four times smaller than `Magma`'s, validating our hypothesis. For the overall algorithm, our total running time is comparable to `Magma`'s for the case $\lambda = \frac{1}{6}$. We also give timings for the case where λ is small. When $\lambda = \frac{1}{12}$ the total running time of our implementation is significantly smaller than `Magma`'s.

1.4 Structure

In Chapter 2 we discuss the necessary background, for example bounds on the set of prime ideals generating the class group. We discuss the `polredabs` algorithm which is designed to produce an f with small roots.

In Chapter 3 we describe the prior work on index-calculus algorithms and the relation generation stage. In Section 3.1 we discuss the index-calculus framework for computing the class group and unit group. Optimizations of index-calculus such as the large prime variation and various approaches to the linear algebra stage are also discussed. In Section 3.1.2 we give an account of the relevant approaches to relation generation. The SIQS approach of Jacobson is described in Sections 3.5.2 – 3.5.3. The previous methods for sieving in number fields of degree larger than two is described in Section 3.6.

Chapter 4 describes the contributions we have discussed in Section 1.2 and Section 1.3. In Chapter 5 we discuss the details of our new implementation of the complete index-calculus framework. Chapter 6 discusses our computational results. In Chapter 7 we give the conclusion and discuss directions for future work.

Chapter 2

Background

Contents

This chapter introduces the mathematical tools employed throughout the remainder of this thesis. A large part of the material is derived from chapters 2 to 4 of [Coh93]. Another important reference is [Bel04]. This paper treats many of the topics in [Coh93] and gives improved algorithms, especially regarding [Coh93, Chapters 2–4].

2.1 Polynomials

Polynomials Over Q

Let $f = \sum_{i=0}^{d} a_i X^i$ be a polynomial with coefficients in $\mathbf{Q}$. It is useful to have upper bounds for the largest absolute value of the complex roots of f in terms of its coefficients. Two bounds ([Cau91, Section v]) are

$$|r| \leq 1 + \max\left\{\left|\frac{a_{d-1}}{a_d}\right|, \left|\frac{a_{d-2}}{a_d}\right|, \ldots, \left|\frac{a_1}{a_d}\right|, \left|\frac{a_0}{a_d}\right|\right\}. \tag{2.1}$$

and

$$|r| \leq 2 \max\left\{\left|\frac{a_{d-1}}{a_d}\right|, \left|\frac{a_{d-2}}{a_d}\right|^{\frac{1}{2}}, \ldots, \left|\frac{a_1}{a_d}\right|^{\frac{1}{(d-1)}}, \left|\frac{a_0}{a_d}\right|^{\frac{1}{d}}\right\}.$$

where r represents any complex root of f.

The **height** of a polynomial is defined as $H(f) = \max\{|a_0|, \ldots, |a_d|\}$. The **Mahler measure** of a polynomial is defined as $M(f) = a_d \prod_{i=1}^{d} \max\{1, |r_i|\}$ where r_i are the complex roots of f.

Resultant

We consider the polynomial ring $R = \mathbf{Z}[X_0, \ldots, X_{k-1}]$ for indeterminates $X_0, \ldots, X_{k-1}$. Let $\mathbf{K}$ be the quotient field of R. Let f and g be polynomials in $R[Z]$ with factorizations

$$f = a(Z - r_0) \cdots (Z - r_{d-1}) \text{ and}$$
$$g = b(Z - s_0) \cdots (Z - s_{n-1})$$

over the algebraic closure $\overline{\mathbf{K}}$ of $\mathbf{K}$. Their resultant is defined

$$\mathrm{Res}_Z(f, g) = a^n \cdot b^d \prod_{\substack{0 \leq i \leq d-1 \\ 0 \leq j \leq n-1}} (r_i - s_j).$$

The resultant has coefficients in R.

Properties of the resultant include:

- For a constant $f = a$, $\operatorname{Res}_Z(a, g) = a^n$.
- Symmetry: $|\operatorname{Res}_Z(f, g)| = |\operatorname{Res}_Z(g, f)|$.
- Multiplicativity: $\operatorname{Res}_Z(f \cdot h, g) = \operatorname{Res}_Z(f, g) \cdot \operatorname{Res}_Z(h, g)$.

The resultant may also be expressed in terms of the roots of one of the polynomials.

$$\operatorname{Res}_Z(f, g) = (-1)^{d \cdot n} \cdot b^d \prod_{i=0}^{n-1} f(s_i) \tag{2.2}$$

One approach to computing the resultant is through the Sylvester matrix of the polynomials. If the coefficients of f are $\{a_i\}$ and the coefficients of g are $\{b_i\}$ the Sylvester matrix is defined:

$$S = \begin{pmatrix} a_d & a_{d-1} & \cdots & a_0 & 0 & \cdots & 0 \\ 0 & a_d & a_{d-1} & \cdots & a_0 & \cdots & 0 \\ \vdots & \ddots & \ddots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & a_d & a_{d-1} & \cdots & a_0 \\ b_m & b_{m-1} & \cdots & b_0 & 0 & \cdots & 0 \\ 0 & b_m & b_{m-1} & \cdots & b_0 & \cdots & 0 \\ \vdots & \ddots & \ddots & & \vdots & \ddots & \vdots \\ 0 & \cdots & 0 & b_m & b_{m-1} & \cdots & b_0 \end{pmatrix}$$

The resultant $\operatorname{Res}_Z(f, g)$ is equal to $\det S$ (see [Coh93, Section 3.3]). This is the approach used by [Coh93] for computing resultants.

Example 2.1. *We let $R = \mathbf{Z}[X_0]$ and define two polynomials in $R[Z]$:*

1. $f(X_0, Z) = X_0^2 + X_0 Z + Z^2$,
2. $g(X_0, Z) = -X_0 + 2X_0 Z + 3Z^2$.

For f, we have $a_2 = 1$, $a_1 = X_0$, and $a_0 = X_0^2$. For g, $b_2 = 3$, $b_1 = 2X_0$, and $b_0 = -X_0$. The

Sylvester matrix is constructed as follows:

$$\begin{pmatrix} 1 & X_0 & X_0^2 & 0 \\ 0 & 1 & X_0 & X_0^2 \\ 3 & 2X_0 & -X_0 & 0 \\ 0 & 3 & 2X_0 & -X_0 \end{pmatrix}$$

We have that $\mathrm{Res}_Z(f, g)$ *is the determinant of this matrix. It is clearly a polynomial in* R. *The determinant computation can be performed by any method, for example, cofactor expansion. It is* $\mathrm{Res}_Z(f, g) = X_0^2 \cdot (7X_0^2 + 5X_0 + 1)$.

As another way to obtain this result we can use equation (2.2)*. We observe that* $f(X_0, 1)$ *is the 3rd cyclotomic polynomial and so over the complex numbers we may factor* f *as* $f = (Z - X_0\omega)(Z - X_0\omega^2)$ *where* ω *is a primitive cube root of unity. Thus*

$$\begin{aligned}\mathrm{Res}_Z(f, g) &= g(X_0\omega) \cdot g(X_0\omega^2) \\ &= 9X_0^4\omega^6 + 6X_0^4\omega^5 + (6X_0^4 - 3X_0^3)\omega^4 + 4X_0^4\omega^3 - 5X_0^3\omega^2 - 2X_0^3\omega + X_0^2.\end{aligned}$$

After simplifying with $\omega^2 + \omega + 1 = 0$ *we get the same result as before.*

The **discriminant** of a degree d polynomial f with leading coefficient c_d is defined

$$\mathrm{disc}(f) = (-1)^{d(d-1)/2} c_d^{-1} \,\mathrm{Res}_Z(f, f'). \tag{2.3}$$

2.2 Linear Algebra and Lattices

2.2.1 Linear Algebra

Two matrix normal forms are commonly employed in the linear algebra phase of class group algorithm algorithms.

The **Hermite Normal Form** (HNF) of an $m \times n$ integer matrix M is denoted by $\mathrm{HNF}(M) = [0|H]$, where H possesses the following characteristics: it is upper triangular, its pivot elements are positive and situated below those in the preceding rows, and the elements to the right of a pivot are nonnegative and strictly smaller than the pivot itself.

Consider A as an $m \times n$ matrix with integer entries. It is guaranteed that there exists a matrix M in HNF, represented as $M = AU$, where U is an invertible matrix with integer entries (see [Coh93, Theorem 2.4.3]).

An $n \times n$ integer matrix M is said to be in **Smith Normal Form** (SNF) if it is diagonal and the diagonal element $m_{i+1,i+1}$ divides $m_{i,i}$ for all $i < n$. Refer to [Coh93, Definition 2.4.11] for more details.

For any $n \times n$ integer matrix A with a nonzero determinant, a unique matrix in Smith Normal Form M exists, such that $M = VAU$, where both U and V are invertible matrices with integer entries (see [Coh93, Theorem 2.4.12]).

For class group algorithms we only need to obtain the structure of the class group using the Smith Normal Form. One way to obtain this is to first compute the Hermite Normal Form, $\text{HNF}(M) = [0|H]$. We then consider the $k \times k$ submatrix H' of H in which the diagonal entries are larger than 1. By permuting the rows and columns of H', we obtain a matrix in Smith Normal Form.

2.2.2 Lattices

One way to add structure to a free $\mathbf{Z}$-module is by assigning lengths to vectors using generalizations of the dot product $\langle \mathbf{x}, \mathbf{y} \rangle = \mathbf{x} \cdot \mathbf{y} = \sum_{i \leq d} x_i y_i$ and L_2 norm $|| \cdot ||_2$.

A **bilinear form** $b : V \times V \to \mathbf{R}$ for V an $\mathbf{R}$-vectorspace satisfies $b(x + x', y) = b(x, y) + b(x', y)$ and $b(\lambda \cdot x, y) = \lambda b(x, y)$ for all $\lambda \in \mathbf{R}, x, x'$ and y in V.

A map q from V to $\mathbf{R}$ is a **positive-definite quadratic form** if $q(\lambda \cdot x) = \lambda^2 q(x)$ for every $\lambda \in \mathbf{R}$ and $\mathbf{x} \in V$ and q is positive-definite. We have that $b(x, y) = \frac{1}{2}(q(x + y) - q(x) - q(y))$ is a (symmetric) bilinear form. Let b be such a form and Λ be a free-$\mathbf{Z}$-module with basis $(\mathbf{b}_i)_{i \leq n}$. Then for any $x \in \Lambda$ we have $\mathbf{x} = \sum_{1 \leq i \leq n} x_i \mathbf{b}_i$ with $x_i \in \mathbf{Z}$. It can be shown that $q(\mathbf{x}) = \sum_{1 \leq i,j \leq n} q_{i,j} x_i x_j$ with $q_{i,j} = b(\mathbf{b}_i, \mathbf{b}_j)$ and we let Q be the matrix $Q = [q_{i,j}]$. This matrix is called the Gram matrix of q.

For convenience, we use

$$||\mathbf{y}|| = q(\mathbf{y})^{1/2}$$

for any $\mathbf{y} \in V$.

A pair $(\mathcal{L}, q)$ with $\mathcal{L}$ a free-$\mathbf{Z}$-module of finite rank m and q a positive-definite quadratic form on $L \otimes \mathbf{R}$ is a **lattice**. In this case the matrix Q defined above is symmetric and positive-definite. The

determinant of $(\mathcal{L}, q)$ is defined

$$\det(\mathcal{L}) = \sqrt{\det(B^t B)} = |\det B|$$

where B is the $m \times m$ matrix whose rows give a basis of the lattice on the standard basis of $L \otimes \mathbf{R}$. The basis matrix B is related to Q by $Q = B^t B$.

The rank of a lattice $(\mathcal{L}, q)$ is the rank of $\mathcal{L}$, and it is said to have full rank if $\mathcal{L}$ has full rank. The **index** of the sublattice $(\mathcal{L}', q)$ in $(\mathcal{L}, q)$ is defined to be the **Z**-module index $[\mathcal{L} : \mathcal{L}']$. We have

$$[\mathcal{L} : \mathcal{L}'] = \frac{\det(\mathcal{L}')}{\det(\mathcal{L})} \tag{2.4}$$

[PZ89, Lemma 3.6].

In a lattice $(\mathcal{L}, q)$ of full rank there is a tuple of nonnegative real numbers $(\lambda_i)_{i \leq m}$ that are the smallest such numbers with the property that there exist m linearly independent vectors $(\mathbf{v}_i)_{i \leq m}$ in $(\mathcal{L}, q)$ satisfying $q(\mathbf{v}_j) \leq \lambda_i$ for all $1 \leq j \leq i$. We call λ_i the ith **successive minimum** of $(\mathcal{L}, q)$ [PZ89, Definition 3.29]. In particular, $\mathbf{v}_1$ is called a **shortest vector**, and the problem of finding such a $\mathbf{v}_1$ is the **shortest vector problem**. The λ_i satisfy

$$\prod_{i \leq m} \lambda_i \leq \mu_m^m \det(\mathcal{L})^2 \text{ and, consequently,}$$
$$\lambda_1 \leq \mu_m \det(\mathcal{L})^{\frac{2}{m}}. \tag{2.5}$$

for a constant μ_m that depends only on the rank m of $\mathcal{L}$ [PZ89, Theorem 3.34].

Finding Short Vectors

The shortest vector in a rank-d lattice $(\mathcal{L}, q)$ can be found using the method of **enumeration**. This type of algorithm takes input a $d \times d$ real number matrix Q defining q and a positive constant c. It outputs all nonzero vectors $\mathbf{x} \in \mathbf{Z}^d$ such that $q(\mathbf{x}) \leq c$. A version of an enumeration algorithm of Fincke and Pohst is given in [Coh93, Algorithm 2.7.7]. Although enumeration algorithms take exponential time in d they are efficient for small dimension.

Another way to find short vectors in a rank-d lattice $(\mathcal{L}, q)$ is to use **lattice basis reduction**. The LLL algorithm [LHLL82] takes as input a basis for a lattice and outputs a reduced basis with shorter, nearly orthogonal vectors. LLL uses a parameter δ such that $\frac{1}{4} < \delta < 1$. Larger δ means the resulting basis will have smaller elements but the running time will be higher [Bel04]. The lattice basis $(\mathbf{b})_{i \leq d}$ is called **δ-LLL-reduced** with respect to q if $\delta \in (\frac{1}{4}, 1]$ and two properties are satisfied

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

for $j < i \leq d$ and

$$||\mathbf{b}_i^* + \mu_{i,i-1}\mathbf{b}_{i-1}^*||^2 \geq \delta \cdot ||\mathbf{b}_{i-1}^*||^2$$

for $1 < i \leq d$. Here $\mathbf{b}_i^*$ are an orthogonal basis (of the ambient space V) defined recursively by $\mathbf{b}_i^* = \mathbf{b}_i - \sum_{j=1}^{i-1} \mu_{i,j}\mathbf{b}_j^*$ for $i \leq d$ and $\mu_{i,j} = \langle \mathbf{b}_i \cdot \mathbf{b}_j^* \rangle / \langle \mathbf{b}_j^* \cdot \mathbf{b}_j^* \rangle$ for $j < i \leq d$.

The constant

$$c_\delta = \frac{1}{(\delta - \frac{1}{4})^{d(d-1)/4}}$$

appears in LLL bounds we will state later.

2.3 Number Fields

Representation of Fields and Field Elements

A number field $\mathbf{K}$ is an extension of the rational numbers of the form $\mathbf{Q}[X]/(f)$ where f is an irreducible polynomial over the rationals. The degree of $\mathbf{K}$ is defined as the degree of f.

Our definition emphasizes the representation of $\mathbf{K}$ as this is an important topic in the sequel. This definition is equivalent to defining a number field as a finite extension of the rational numbers due to the primitive element theorem. This theorem tell us that any finite extension of the rational numbers is generated over $\mathbf{Q}$ by a single algebraic number θ, a root of an irreducible polynomial f. Furthermore, θ can be chosen to be an algebraic integer and, consequently, f will be monic. In the sequel, we always obtain a monic defining polynomial f for $\mathbf{K}$ as a precomputation and do all subsequent computations with $\mathbf{K}$ represented as $\mathbf{Q}[X]/(f)$. Given a number field $\mathbf{K} = \mathbf{Q}[X]/(f)$ we use θ to denote an element such that $\mathbf{K} = \mathbf{Q}(\theta)$.

We call f a **defining polynomial** for $\mathbf{K}$ if it is the minimal polynomial of a primitive element of $\mathbf{K}$. We note that if f is a defining polynomial then so is $f(X + a)$ for any integer a. Thus there are infinitely many defining polynomials.

Representations of elements of $\mathbf{K}$ and arithmetic with these elements are discussed in [Coh93, Section 4.2.].

Signature and Embeddings

Computing the d roots of f in $\mathbf{C}$ can be done efficiently using in [Coh93, Algorithm 3.6.6]. The **signature** of $\mathbf{K}$ is (r_1, r_2) where r_1 is the number of real roots and r_2 is the number of pairs of complex conjugate roots. We denote the complex conjugate of α by $\overline{\alpha}$. As complex roots come in conjugate pairs we have $d = r_1 + 2r_2$.

As f has d roots in $\mathbf{C}$ there are d embeddings $(\sigma_i)_{i \leq n}$ of the $\mathbf{K}$ into $\mathbf{C}$. Embeddings of $\mathbf{K}$ into $\mathbf{R}^{r_1} \times \mathbf{C}^{r_2}$ and $\mathbf{R}^d$ can now be constructed using the $(\sigma_i)_{i \leq d}$ [Coh93, Section 4.2.4]. We can order the $(\sigma_i)_{i \leq r_1 + 2r_2}$ so that $\sigma_i = \overline{\sigma_{i+r_2}}$ for all $i > r_1 + r_2$.

One choice of embedding of $\mathbf{K}$ is $\tau : \mathbf{K} \to \mathbf{R}^{r_1} \times \mathbf{C}^{r_2}$ defined

$$\tau(x) = (\sigma_1 x, \ldots, \sigma_{r_1+r_2} x)\,.$$

A second embedding $\sigma : \mathbf{K} \to \mathbf{R}^d$ is defined

$$\sigma(x) = \Big(\sigma_1 x, \ldots, \sigma_{r_1} x, \sqrt{2}\mathrm{Re}(\sigma_{r_1+1}x), \sqrt{2}\mathrm{Im}(\sigma_{r_1+1}x), \ldots \sqrt{2}\mathrm{Re}(\sigma_{r_1+r_2}x), \sqrt{2}\mathrm{Im}(\sigma_{r_1+r_2}x)\Big)$$

where Re is the real part and Im is the complex part. Weighting by a factor of $\sqrt{2}$ is done so that $||\sigma(x)||_2 = ||\tau(x)||_2$.

Another important type of embedding is the **logarithmic embedding** denoted Log [Coh93, 4.9.6]. Again we use the embeddings σ_i of $\mathbf{K}$ into $\mathbf{C}$ but this time we embed $\mathbf{K}^*$ into $\mathbf{R}^{r_1+r_2}$. It is defined

$$\operatorname{Log} x = (n_i \log |\sigma_i x|)_{i \leq r_1 + r_2}$$

where $n_i = 1$ for $i \leq r_1$ and $n_i = 2$ otherwise.

Analytic Norms for Field Elements

With the embeddings σ_i we can define analytic norms for number field elements. The most well-known norm is

$$||x|| = ||\sigma(x)||_2 = \left(\sum_{i=1} |\sigma_i x|^2\right)^{\frac{1}{2}}$$

for all $x \in \mathbf{K}$. Another norm introduced by [Buc90] is

$$||x||_{\mathbf{v}} = \left(\sum_{i=1}^{d} e^{v_i} |\sigma_i x|\right)^{\frac{1}{2}} \tag{2.6}$$

for all $x \in \mathbf{K}$ and for a vector of real numbers $\mathbf{v} = (v_i)_{i \le d}$ such that $v_{r_2+i} = v_i$ for $r_1 < i \le r_1 + r_2$. In [Coh93, Section 6.5] it is named the **v**-norm.

Discriminant and Integral Basis

The algebraic integers of $\mathbf{K} = \mathbf{Q}(\theta)$ are elements of $\mathbf{K}$ that have a monic minimal polynomial with integer coefficients. The **ring of algebraic integers** is the ring these elements generate and is denoted $\mathcal{O}_{\mathbf{K}}$. This ring is a free $\mathbf{Z}$-module of rank d [Coh93, Theorem 4.4.2]. An **integral basis** of $\mathbf{K}$ is a $\mathbf{Z}$-basis of the free module $\mathcal{O}_{\mathbf{K}}$.

The discriminant $|\Delta|$ is an invariant of the number field that can be defined using an integral basis $\omega_0, \ldots, \omega_{d-1}$. The discriminant is defined as the determinant of the lattice generated by the ω_i under σ. In the context of computing class groups, the absolute value of $|\Delta|$ is used as a measure of the size of the number field.

The field discriminant is related to the discriminant of the defining polynomial disc(f) by

$$|\Delta| = \text{disc}(f)[\mathcal{O}_{\mathbf{K}} : \mathbf{Z}[\theta]]^{-2}$$

(see [Coh93, Theorem 4.4.4]).

We call $[\mathcal{O}_{\mathbf{K}} : \mathbf{Z}[\theta]]$ the **index** of θ. Unlike $|\Delta|$, the index depends on the choice of f.

No efficient algorithm for computing an integral basis of $\mathcal{O}_{\mathbf{K}}$ for an arbitrary number field of degree d is known. However, there is an efficient algorithm that solves this problem if we are given $|\Delta|$ and the prime factorization of $|\Delta|$. The integral bases can then be computed using the **Round 2 algorithm**. A full description of this algorithm, which involves computing successive "p-radicals" of certain ideals, is described in [Coh93, Section 6.1].

Roots of Unity

The roots of unity in any number field form a cyclic group $\langle \omega_m \rangle$ of order m where $m \mid d$. The way to compute the roots of unity suggested by [Coh93, Algorithm 4.9.9] is to use Fincke-Pohst enumeration.

In a degree four number field, m is one of the solutions to $\phi(x) \mid 4$ where ϕ is the Euler totient function. For $\phi(x) = 2$ the possible solutions are $m = 3, 4, 6$ and for $\phi(x) = 4$ the possible solutions are $m = 5, 8, 10$ and 12.

Conditions for Galois groups and subfields

For quartic fields $\mathbf{K}$ there are five possibilities for the Galois group:

- S_4 the symmetric group of order 24,
- A_4 the alternating group of order 12,
- D_4 the dihedral group of order 8 ($\mathbf{K}$ is called a **dihedral** extension),
- V_4 the Klein four-group of order 4 ($\mathbf{K}$ is called a **biquadratic** extension),
- C_4 the cyclic group of order 4 ($\mathbf{K}$ is called a **cyclic** extension).

A field is called **primitive** if it has no nontrivial subfields. For degree four fields the only possible degree of a nontrivial subfield is 2 as the degree of a subfield must divide 4. We have $\mathbf{K}$ is nonprimitive if and only if G is one of D_4, V_4, C_4 [HSW95].

If $\mathbf{K}$ contains a quadratic subfield $\mathbf{Q}(\sqrt{c})$ for c a squarefree integer then $\mathbf{K}$ can be represented as $\mathbf{K} = \mathbf{Q}(\sqrt{c}, \sqrt{a + b\sqrt{c}})$, or defined by the minimal polynomial of $\sqrt{a + b\sqrt{c}}$ which is $f = X^4 + 2aX^2 + (a^2 - cb^2)$, where a, b, c are integers and both c and $\gcd(b, c)$ are squarefree. A full characteriziation of the Galois groups of such a field is given by [KW89] and [HSW95]:

$$\mathbf{K} \text{ is biquadratic} \Leftrightarrow a^2 - cb^2 = k^2 \text{ for some integer } k.$$

$$\mathbf{K} \text{ is cyclic} \Leftrightarrow a^2 - cb^2 = ck^2 \text{ for some integer } k.$$

$$\mathbf{K} \text{ is dihedral} \Leftrightarrow \mathbf{K} \text{ is not biquadratic and } \mathbf{K} \text{ is not cyclic.}$$

A possible use of this characterization is to construct quartic defining polynomials with particular Galois groups of their splitting fields.

Irrespective of how the field is represented, there are algorithms for computing the Galois group that are efficient in practice for degree four e.g. [Coh93, Algorithm 6.3.3].

2.4 Ideals

An **integral ideal $\mathfrak{a}$** of $\mathcal{O}_{\mathbf{K}}$ is a sub-$\mathcal{O}_{\mathbf{K}}$-module of $\mathcal{O}_{\mathbf{K}}$, i.e., a sub-$\mathbf{Z}$-module of $\mathcal{O}_{\mathbf{K}}$ such that for every $x \in \mathcal{O}_{\mathbf{K}}$ and $y \in \mathfrak{a}$ we have $xy \in \mathfrak{a}$. A **ideal $\mathfrak{a}$** is a nonzero sub-$\mathbf{Z}$-module of $\mathbf{K}$ so that there exists a nonzero integer a with $a\mathfrak{a}$ an ideal of $\mathcal{O}_{\mathbf{K}}$. The term "ideal" in the sequel refers to a fractional ideal. The smallest such a is called the **denominator**.

We may write a ideal as a quotient of integral ideals. In particular, given any ideal $\mathfrak{a}$ we can represent $\mathfrak{a}$ as the pair $(\mathfrak{b}, a)$ such that $\mathfrak{a} = \mathfrak{b}/a$ for $\mathfrak{b}$ a integral ideal and a the denominator of $\mathfrak{b}$. This representation is useful for two reasons. First, multiplication of ideals can be done pointwise on elements of the pair. Second, retaining the denominator a is useful for many algorithms that work with ideals (see [Bel04]).

We use $(x_i)_{i=0}^{k-1}$ to denote the ideal $x_0\mathcal{O}_{\mathbf{K}} + \cdots + x_{k-1}\mathcal{O}_{\mathbf{K}}$. When $k = 1$ the ideal is **principal**.

Ideal multiplication is defined as

$$\mathfrak{a}\mathfrak{b} = \left\{ \sum_{i=1}^{k} x_i y_j : k \in \mathbf{N}, x_i \in \mathfrak{a}, y_i \in \mathfrak{b} \right\}.$$

For any ideal $\mathfrak{a}$ of $\mathcal{O}_{\mathbf{K}}$ there is a unique ideal **inverse** so that $\mathfrak{a}\mathfrak{a}^{-1} = \mathcal{O}_{\mathbf{K}} = (1)$ defined $\mathfrak{a}^{-1} = \{x \in \mathbf{K}, x\mathfrak{a} \subseteq \mathcal{O}_{\mathbf{K}}\}$. The set of all ideals $\mathcal{I}$ is a group under multiplication. Similarly, the set of all principal ideals $\mathcal{P}$ is a group under multiplication (see [Coh93, Theorem 4.6.14]).

A prime ideal $\mathfrak{p}$ in a ring R is defined as follows: $\mathfrak{p}$ is prime if $\mathfrak{p} \neq R$ and the quotient ring $R/\mathfrak{p}$ is an integral domain. That is, the ideal $\mathfrak{p}$ satisfies two conditions: $\mathfrak{p} \neq R$ and for any elements $a, b \in R$, if their product $ab \in \mathfrak{p}$, then $a \in \mathfrak{p}$ or $b \in \mathfrak{p}$. Equivalently, the quotient ring $R/\mathfrak{p}$ does not contain zero divisors.

If p is a prime in $\mathbf{Q}$ then there exist positive integers g and $(e_i)_{i=0}^{g-1}$ such that

$$(p) = \prod_{i=0}^{g-1} \mathfrak{p}_i^{e_i}$$

where the $\mathfrak{p}_i$ are the prime ideals of $\mathbf{K}$ so that $\mathfrak{p}_i \cap \mathbf{Z} = p$. These are called the primes **above** p. The **ramification index** of p at $\mathfrak{p}_i$ is the integer e_i. The **degree** f_i of $\mathfrak{p}_i$ is the degree of the finite field $\mathcal{O}_{\mathbf{K}}/\mathfrak{p}_i$ over $\mathbf{Z}/p\mathbf{Z}$. We have $\sum_{i=0}^{g-1} e_i f_i = d$. Prime ideals will be discussed further in Section 2.4.1.

An ideal $\mathfrak{a}$ can be represented as a matrix A of column vectors $(\mathbf{b}_i)_{i \le d}$ giving the coordinates of the elements of a $\mathbf{Z}$-basis of $\mathfrak{a}$ in terms of a $\mathbf{Z}$-basis of $\mathcal{O}_{\mathbf{K}}$. The HNF of A gives a unique $\mathbf{Z}$-basis representation.

The operations of ideal multiplication and inversion have a role in some index-calculus class group algorithms. Efficient algorithms for these operations are given in [Bel04, Section 5.3, 5.4].

Theorem 2.2 (Chinese Remainder Theorem)**.** *Let $\mathfrak{a}_1, \ldots, \mathfrak{a}_n$ be ideals in $\mathcal{O}_{\mathbf{K}}$ such that for every pair $i \neq j$, $\mathfrak{a}_i + \mathfrak{a}_j = \mathcal{O}_{\mathbf{K}}$. Then the map*

$$\begin{aligned} \mathcal{O}_{\mathbf{K}} / \left(\cap_{i=1}^{n} \mathfrak{a}_i \right) &\longrightarrow \mathcal{O}_{\mathbf{K}}/\mathfrak{a}_1 \times \cdots \times \mathcal{O}_{\mathbf{K}}/\mathfrak{a}_n \\ a &\longmapsto (a + \mathfrak{a}_1, \ldots, a + \mathfrak{a}_n) \end{aligned}$$

is an isomorphism.

The **norm** of a nonzero ideal $\mathfrak{a}$ is the number of elements in $\mathcal{O}_{\mathbf{K}}/\mathfrak{a}$:

$$N(\mathfrak{a}) = |\mathcal{O}_{\mathbf{K}}/\mathfrak{a}|.$$

Theorem 2.3. *For two ideals $\mathfrak{a}, \mathfrak{b}$ of $\mathcal{O}_{\mathbf{K}}$ we have $N(\mathfrak{a} \cdot \mathfrak{b}) = N(\mathfrak{a}) \cdot N(\mathfrak{b})$.*

This theorem is useful when we consider multiplicative decompositions of an ideal. For instance, we can decompose an ideal into its prime factors or, if the ideal is principal, it might be decomposed into a product of principal ideals. The following facts can be used for these respective cases:

- The norm of a prime ideal $\mathfrak{p}$ above p is p^f where f is the degree of $\mathfrak{p}$.
- The norm of a principal ideal $\mathfrak{a} = (\alpha)$ is the absolute value of the norm of α.

The first tells us that the prime factorization of an ideal $\mathfrak{a}$ determines the factorization of its norm. The second tells us that if we are given $(\beta) = (\prod \alpha_i)$ then the $N((\alpha_i))$ determine $N((\beta))$. Norms of principal ideals (i.e. norms of elements of $\mathbf{K}$) are the ones we are primarily interested in. These are discussed in Section 2.5.

Given a ideal $\mathfrak{a}$ with $\mathbf{Z}$-basis $(\mathbf{b}_i)_{i\leq d}$ we use $\sigma(\mathfrak{a})$ to denote the lattice generated by the image of $\mathbf{b}_i$ under σ (recalling that σ is one of the embeddings we have defined). Using $N(\mathfrak{a}) = [\sigma(\mathcal{O}_{\mathbf{K}}) : \sigma(\mathfrak{a})]$ in equation (2.4) we can compute the determinant of $\sigma(\mathfrak{a})$.

Lemma 2.4. *Let $\mathfrak{a}$ be an integral ideal of $\mathbf{K}$, then $\sigma(\mathfrak{a})$ is a sublattice of $\mathbf{R}^d$ and $\det(\sigma(\mathfrak{a})) = |\Delta|^{1/2}N(\mathfrak{a})$.*

A bound on $||\omega||$ for ω an element of an LLL-reduced basis of $\mathcal{O}_{\mathbf{K}}$ follows from basic properties of LLL (see [Bel04, Proposition 5.1]). More generally, we can apply the same idea to any integral ideal using Lemma 2.4.

Theorem 2.5. *If $w_1, \dots, w_d$ is an LLL-reduced basis of an integral ideal $\mathfrak{a}$ with respect to $||\cdot||$ then*

$$||w_i|| \leq c_\delta (d^{-\frac{(i-1)}{2}} |\Delta|^{\frac{1}{2}} N(\mathfrak{a}))^{\frac{1}{d-i+1}}$$

for $1 \leq i \leq d$.

2.4.1 Prime Ideals

Decomposition of Prime Numbers

We discuss the factorization of the ideal generated by a rational prime p in $\mathcal{O}_{\mathbf{K}}$. As before f is a monic irreducible defining polynomial for the number field. We now consider the factorization of the ideal generated by a rational prime. The Dedekind-Kummer theorem gives us a way to factor most rational prime ideals into prime ideals in $\mathbf{K}$.

Theorem 2.6. *Let p be a prime not dividing $[\mathcal{O}_{\mathbf{K}} : \mathbf{Z}[\theta]]$, let $f = \prod_{i=1}^{g} f_i^{e_i}$ be the factorization of f into monic irreducible polynomials in the finite field $\mathbb{F}_p$, and let $\overline{f_i}$ be a lift of f_i to $\mathbf{Z}[X]$. Then*

$$(p) = \prod_{i=1}^{g} \mathfrak{a}_i \textit{ where } \mathfrak{a}_i = (p, \overline{f_i(\theta)}).$$

All the $\mathfrak{a}_i$ are prime ideals with $N(\mathfrak{a}_i) = p^{e_i n_i}$ where n_i is the degree of f_i.

We call the pair (p, f_i) the **Dedekind-Kummer** representation of $\mathfrak{p}$. Hence the problem of factoring a prime not dividing the index reduces to factoring a polynomial over a finite field. As discussed in [Coh93, Chapter 3] there are very efficient algorithms for this problem.

When p divides the index, factoring p requires an integral basis of $\mathcal{O}_{\mathbf{K}}$, the computation of which we have already discussed. It is done with a sophisticated algorithm described in [Coh93, Algorithm 6.2.9]. This algorithm is implemented in `PARI/GP` using the `idealprimedec` function.

Valuation at a Prime Ideal

Recall that each fractional ideal $\mathfrak{a}$ can be expressed uniquely as a product $\mathfrak{a} = \prod_p \mathfrak{p}^{\nu_{\mathfrak{p}}(\mathfrak{a})}$, with the product over prime ideals. The exponent $\nu_{\mathfrak{p}}(\mathfrak{a})$ is called the **valuation** of $\mathfrak{a}$ at $\mathfrak{p}$.

The valuation of a nonzero element α in $\mathbf{K}$ at a prime ideal $\mathfrak{p}$ is defined as the highest power n such that α belongs to $\mathfrak{p}^n$ but not to $\mathfrak{p}^{n+1}$. An efficient algorithm for computing the valuation is given in [Bel04, Section 6.1]. The first step is finding an "anti-uniformizer" τ for $\mathfrak{p}$ after which we iteratively multiply $\mathfrak{a}$ by τ until the result is no longer divisible by $\mathfrak{p}$.

The valuation of a nonzero ideal $\mathfrak{a}$ of $\mathcal{O}_{\mathbf{K}}$ at $\mathfrak{p}$ represented as a $\mathbf{Z}$-basis $\alpha_1, \ldots, \alpha_d$ is the minimum of the valuations of the α_i (see [Bel04, Section 6.1]).

Chebotarev Density Theorem

The Chebotarev Density Theorem provides insight into the distribution of prime ideals in Galois extensions. Our description is based on [SL96].

Let $\mathbf{L}/\mathbf{K}$ be a Galois extension with Galois group $G = \mathrm{Gal}(\mathbf{L}/\mathbf{K})$. For a prime ideal $\mathfrak{p}$ of $\mathbf{K}$ which does not divide $\Delta_{\mathbf{L}/\mathbf{K}}$ we choose a prime ideal $\mathfrak{q}$ in $\mathbf{L}$. There is a unique element $\mathrm{Frob}_{\mathfrak{q}}$ in G such that for all α in the maximal order of $\mathbf{L}$ we have $\mathrm{Frob}_{\mathfrak{q}}(\alpha) \equiv \alpha^{|\mathcal{O}_{\mathbf{K}}/p|} \pmod{\mathfrak{q}}$.

The **Frobenius symbol** of $\mathfrak{p}$ in $\mathbf{L}/\mathbf{K}$ is the conjugacy class $\{\mathrm{Frob}_{\mathfrak{q}} : \mathfrak{q}|\mathfrak{p}\}$. We let $\sigma_{\mathfrak{p}}$ denote an arbitrary element in this conjugacy class. For S a set of primes of $\mathbf{K}$ the **natural density** of S is defined as

$$d(S) = \lim_{x\to\infty} \frac{|\{\mathfrak{p} : |\mathcal{O}_{\mathbf{K}}/\mathfrak{p}| \le x, \mathfrak{p} \in S\}|}{|\{\mathfrak{p} : |\mathcal{O}_{\mathbf{K}}/\mathfrak{p}| \le x, \mathfrak{p} \text{ prime}\}|}.$$

Theorem 2.7. *Let $\mathbf{K} \subseteq \mathbf{L}$ be Galois, and let $C \subseteq G = Gal(\mathbf{L}/\mathbf{K})$ be a conjugacy class. Then*

$$\{\mathfrak{p} : \mathfrak{p} \textit{ a prime of } \mathbf{K}, \mathfrak{p} \nmid \Delta_{\mathbf{L}/\mathbf{K}}, \sigma_{\mathfrak{p}} \in C\}$$

has natural density $\frac{\#C}{\#G}$.

The **cycle type** of a permutation in a permutation group is the tuple of the number of cycles of each length that are present in the cycle decomposition. From group theory we know that if two permutations have the same cycle type then they are in the same conjugacy class. The prime **decomposition type** of a rational prime p is the tuple of degrees of the prime ideals above p in $\mathbf{K}$. For p not dividing $|\Delta|$ the prime decomposition type is the same as the cycle type of $\sigma_{\mathfrak{p}}$.

Theorem 2.7 can be used to roughly estimate the probability that a random rational prime will have a particular decomposition type. Note that in the sequel we will refer to these estimates only to compare alternative algorithms rather than in our main results.

As an example we consider the density of prime decomposition types occurring in number fields $\mathbf{K}$ of Galois groups D_4 and S_4. The first column of Table 2.1 gives the Galois group of the number field. Regarding the other columns, the first row gives the decomposition type of primes not dividing the discriminant. Below it are the values $\frac{\#C}{\#G}$. For instance 4 corresponds to the prime being inert and $(1,1,1,1)$ corresponds to the prime being fully split. In the sequel we use this table to estimate

Galois Group	4	$1,3$	$2,2$	$1,1,2$	$1,1,1,1$
D_4	$\frac{1}{4}$	0	$\frac{3}{8}$	$\frac{1}{4}$	$\frac{1}{8}$
S_4	$\frac{1}{4}$	$\frac{1}{3}$	$\frac{1}{8}$	$\frac{1}{4}$	$\frac{1}{24}$

Table 2.1: Decomposition Type Densities

the probability that a rational prime p not dividing $|\Delta|$ but otherwise chosen at random will have two distinct degree one primes above it. This corresponds to decomposition types that contain two or more ones. These are $1,1,2$ and $1,1,1,1$. For an S_4 field an estimate for the probability is $\frac{1}{4}+\frac{1}{24}=\frac{7}{24}$.

2.5 Norms, Norm Forms, and Smooths

Norms

The **norm** of an element $\mathrm{N}:\mathbf{K}\to\mathbf{Q}$ can be defined for $\alpha\in\mathbf{K}$ as

$$\mathrm{N}(\alpha)=\mathrm{N}(\sum_{i=0}^{d-1}c_i\theta^i)=\mathrm{Res}_\theta\left(\sum_{i=0}^{d-1}c_i\theta^i, f(\theta)\right)$$

(see [Coh93, Section 4.3]). It can be shown that for $\alpha\in\mathcal{O}_{\mathbf{K}}$ we have $\mathrm{N}(\alpha)\in\mathbf{Z}$. The norm is a multiplicative function. A simple but useful consequence of this is $a\in\mathbf{Z}$ has norm equal to the dth power of a.

We can bound the norm of a function in two ways. The first uses the arithmetic-geometric mean (AM-GM) inequality. Recall that this inequality gives an upper bound for the geometric mean in terms of the arithmetic mean:

$$\left(\prod_{i=1}^{n} x_i\right)^{\frac{1}{n}} \leq \frac{1}{n}\sum_{i=1}^{n} x_i,$$

where $\{x_i\}_{i=1}^{n}$ denotes a set of nonnegative real numbers. The left hand side, the geometric mean, is defined by multiplications of the terms while the right hand side, the arithmetic mean, is defined by additions of the terms. One reason this inequality is important is that it relates the value of a multiplicative function to the value of an additive function.

Number fields $\mathbf{K}$ have a multiplicative norm, and an additive norm $||\cdot||$. The AM-GM inequality relates these in the following way:

Theorem 2.8 (AM-GM Norm Bound)**.** *For any $\alpha \in \mathcal{O}_{\mathbf{K}}$ we have*

$$|N(\alpha)| \leq d^{-\frac{d}{2}}||\alpha||^{d}.$$

A second way is to express the element α as a polynomial g and bound the norm using a bound on the resultant of two polynomials. This polynomial will have degree less than d.

Theorem 2.9 (Coefficient Norm Bound)**.** *Given $\alpha \in \mathcal{O}_{\mathbf{K}}$ and g a degree $m < d$ polynomial with integer coefficients so that $\alpha = g(\theta)$*

$$|N(\alpha)| \leq (d+1)^{m/2}(m+1)^{d/2}H(g)^{d}\cdot H(f)^{m}.$$

Norm Forms

The **norm form** of the t elements $(\omega_0, \ldots, \omega_{t-1})$ of $\mathbf{Q}(\theta)$ is a map from $\mathbf{K}[X_0, \ldots, X_{t-1}]$ to $\mathbf{Q}[X_0, \ldots, X_{t-1}]$ defined

$$\phi(X_0, \ldots, X_{t-1}) = \mathrm{N}(\sum_{i=0}^{t-1} X_i\omega_i) = \mathrm{Res}_{\theta}\left(\sum_{i=0}^{t-1} X_i\omega_i, f(\theta)\right).$$

As there is an efficient algorithm to compute the resultant the norm form can also be computed efficiently.

The norm form is a degree d homogenous polynomial in t variables and so it can be dehomogenized by dividing by any of the X_i raised to the power d. Thus the set of zeroes of $\phi(X_0, \ldots, X_{t-1})$ and $\phi(X_0, \ldots, X_{i-1}, 1, X_{i+1}, \ldots, X_{t-1})$ over a field $\mathbf{E}$ are the same apart from the trivial solution $X_0 = \cdots = X_{t-1} = 0$ of the homogenized form.

We will always be interested in norm forms where the ω are algebraic integers. In this case, the norm form has integer coefficients. As we will discuss in the next chapter, the norm form of two elements ω_0, ω_1 is used when generating relations by sieving. We let $\mathfrak{a}$ be the gcd of the ideals (ω_0) and (ω_1). Throughout the thesis we will always denote the norm of this ideal by

$$A = \mathrm{N}(\mathfrak{a}).$$

As $\mathfrak{a} \mid (x\omega_0 + y\omega_1)$ for any integers x, y the polynomial $A^{-1}\phi(X, Y)$ has integer coefficients. We will use these polynomials frequently throughout the thesis, so we give them a name.

Definition 2.10. Let ω_0, ω_1 be elements of $\mathcal{O}_{\mathbf{K}}$, let $\mathfrak{a}$ be the gcd of the ideals (ω_0) and (ω_1), and let $A = N(\mathfrak{a})$. The **sieving polynomial** of ω_0, ω_1 is

$$\frac{1}{A}\phi(X, Y)$$

where $\phi(X, Y)$ is the norm form of ω_0, ω_1.

Smooths

Rational numbers are called **B-smooth** if they factor over a set of rational primes called a **factor base** that are smaller than B.

We call an ideal **B-smooth** if it factors over a factor base $\mathrm{FB}_{\mathbf{K}}$ of prime ideals of norm less than B. The set of primes below the factor base primes are called the rational factor base, denoted $\mathrm{FB}_{\mathbf{Z}}$.

If an ideal has B-smooth norm it is likely to be B-smooth itself. However, it is not always the case.

Example 2.11. *Let $\mathbf{K}$ be a cubic number field $\mathbf{K}$ so that the Galois group of the splitting field of $\mathbf{K}$ is S_3. If $\mathfrak{p}$ is a prime ideal above 2 of degree 2 then its norm is 4 and hence $\mathfrak{p}$ has 3-smooth norm. However, $N(\mathfrak{p}) = 4 > 3$ and so $\mathfrak{p}$ is not 3-smooth.*

The probability that a principal ideal is B-smooth relates to the efficiency of the algorithm we give in the sequel. Though we do not use this probability directly in the sequel, its relevance makes it worth mentioning briefly.

The Dickman function $\rho(u)$ is defined as $\rho(u) = 1$ for $0 \leq u \leq 1$ and for $u > 1$ the function satisfies the differential equation $\rho'(u) = -\frac{\rho(u-1)}{u}$. The proportion of positive integers up to x that are $x^{1/u}$-smooth asymptotically approaches $\rho(u)$ as x approaches infinity. Thus, for large values of x, ρ can be used to approximate the density of smooth numbers among the integers up to x. Furthermore,

$$\rho(u) \approx e^{-u(\log u)(1+o(1))}$$

when x is large [Gra08].

Now we transition from smooth integers to smooth ideals. Biasse-Fieker in [BF14] consider the likelihood of an ideal being B-smooth. They introduce a heuristic for estimating the smoothness probability of ideals, specifically focusing on principal ideals with norms bounded by x and their y-smoothness. Their heuristic is that the probability $P(x, y)$ that a principal ideal of norm bounded by x is y-smooth satisfies

$$P_{\mathbf{K}}(x, y) = P_{\mathbf{K}}(x, x^{1/u}) \geq e^{-u(\log u)(1+o(1))}$$

for $u = \frac{\log x}{\log y}$. Its qualitative meaning is that the probability of a principal ideal being B-smooth is at least as large as the B-smoothness probability of a random integer.

2.6 Class Group & Unit Group

Let $\mathcal{I}$ be the group of nonzero fractional ideals of $\mathcal{O}_{\mathbf{K}}$ and $\mathcal{P}$ be the subgroup of principal fractional ideals. The class group is $\mathrm{Cl}(\mathcal{O}_{\mathbf{K}}) = \mathcal{I}/\mathcal{P}$ and its order, the **class number**, is h. It is a finite abelian group [Coh93, Theorem 4.9.2]. The ideals $\mathfrak{a}$ and $\mathfrak{b}$ are **equivalent** if there exists a non-zero $\gamma \in \mathbf{K}$ such that $\mathfrak{a} = (\gamma)\mathfrak{b}$ and we write $\mathfrak{a} \sim \mathfrak{b}$.

The group of units of $\mathcal{O}_{\mathbf{K}}$, denoted by $\mathcal{U}(\mathcal{O}_{\mathbf{K}})$, consists of all elements in $\mathcal{O}_{\mathbf{K}}$ that have a multiplicative inverse in $\mathcal{O}_{\mathbf{K}}$. The Dirichlet Unit Theorem tells us that the unit group $\mathcal{U}(\mathcal{O}_{\mathbf{K}})$ is finitely generated and isomorphic to the direct product of the group of roots of unity in the field and

$$r = r_1 + r_2 - 1$$

copies of the infinite cyclic group $\mathbf{Z}$. The rank r of the free part of the group of units is known as the rank of the unit group of $\mathbf{K}$. From this theorem it follows that the lattice generated by the logarithmic embedding Log of the units u has rank r. A set of units $u_1, u_2, \ldots, u_r$ whose images under Log are a basis of this lattice is called a **system of fundamental units**.

Let $u_1, u_2, \ldots, u_r$ be a system of fundamental units. The **regulator** R of $\mathbf{K}$ is defined as the absolute value of the determinant of the $r \times r$ matrix M whose (i, j)-th entry is given by

$$M_{ij} = \begin{cases} \log|\sigma_j(u_i)| & \text{for } 1 \leq j \leq r_1, \\ 2\log|\sigma_j(u_i)| & \text{for } r_1 < j \leq r \end{cases}$$

The regulator R is independent of the choice of the system of fundamental units.

Other representations of a system of fundamental units can also be used. For example in [Thi95] an efficient algorithm for computing a system of fundamental units represented as a power-product of small elements of $\mathcal{O}_{\mathbf{K}}$ can be recovered from this basis.

Analytic Class Number Formula

The **Dedekind zeta function** $\zeta_{\mathbf{K}}$ is defined $\zeta_{\mathbf{K}} = \sum_{\mathfrak{a}} \frac{1}{N(\mathfrak{a})^s}$ where the sum is over the nonzero integral ideals of $\mathcal{O}_{\mathbf{K}}$ (see [Coh93, Section 4.9]). This function is equal to an **Euler product**

$$\zeta_{\mathbf{K}} = \prod_{\mathfrak{p}} \frac{1}{1 - \frac{1}{N(\mathfrak{p})^s}}$$

where the product is over all non-zero prime ideals of $\mathcal{O}_{\mathbf{K}}$.

The **analytic class number formula** connects algebraic invariants of a number field–discriminant Δ, signature (r_1, r_2), number of roots of unity m, class number h, and regulator R–and the analytic properties of its zeta function. We can write it so that it gives the product of the class number and regulator in terms of other quantities.

$$hR = \frac{m\sqrt{|\Delta|}}{2^{r_1}(2\pi)^{r_2}} \lim_{s \to 1}(s-1)\zeta_K(s).$$

We will consider estimating $\lim_{s \to 1}(s-1)\zeta_{\mathbf{K}}(s)$ in the next chapter. As we have discussed, the other quantities on the right hand side can be computed efficiently in practice.

Prime ideals generating the ideal class group

Let χ be a non-principal Dirichlet character modulo q, and let $L(s, \chi)$ be the L-function defined by χ. Then the Generalized Riemann Hypothesis (GRH) asserts that all non-trivial zeros of $L(s, \chi) = \sum_{n=1}^{\infty} \frac{\chi(n)}{n^s}$ lie on the critical line, i.e., for any non-trivial zero ρ of $L(s, \chi)$, we have: $\mathrm{Re}(\rho) = \frac{1}{2}$.

Under GRH Bach [Bac90] shows that for a number field of degree d the set of prime ideals with

$$N(\mathfrak{p}) < 12(\log|\Delta|)^2$$

contains a generating subset of $\mathrm{Cl}(\mathcal{O}_{\mathbf{K}})$. Several variations of this result are possible. In the case of quadratic fields the smaller bound

$$N(\mathfrak{p}) < 6(\log|\Delta|)^2$$

can be used instead. For general degrees and for sufficiently large $|\Delta|$ Bach shows the constant 12 can be decreased to 4. Recently in [GM22] this is improved to

$$N(\mathfrak{p}) < \left(4 - \frac{1}{3d}\right)(\log|\Delta|)^2.$$

In the case of quartic fields this gives $N(\mathfrak{p}) < 3.92(\log|\Delta|)^2$.

In [BDyDF08] it is shown that, under the assumption of GRH, if T satisfies the inequality

$$\sum_{\substack{m \geq 1 \\ N(\mathfrak{p}^m) < T}} \frac{\log N(\mathfrak{p})}{N(\mathfrak{p}^{m/2})} \left(1 - \frac{\log N(\mathfrak{p}^m)}{\log(T)}\right) > \frac{\log|\Delta|}{2} - 1.9d - 0.785r_1 + \frac{2.468 + 1.832r_1}{\log T} \tag{2.7}$$

then the class group of $\mathbf{K}$ is generated by prime ideals of $\mathcal{O}_{\mathbf{K}}$ having norm strictly smaller than T. This sum is over the prime ideals of $\mathfrak{p}$ of $\mathcal{O}_{\mathbf{K}}$ and all positive integers m so that $N(\mathfrak{p}^m) < T$. Sufficiently large T are guaranteed be solutions to (2.7) because the sum diverges as T goes to infinity. If we choose $B > t$ where t is the smallest T satisfying (2.7) then the prime ideals with norm less than B generate the class group. The authors show that t can be computed efficiently compared to the overall cost of running the index-calculus algorithm. They note this approach often improves on the Bach bound by a factor of 20 or more.

2.7 Defining Polynomials

A degree d number field $\mathbf{K}$ can be represented by many different defining polynomials g.

Polynomial reduction algorithms take a defining polynomial g of $\mathbf{K} = \mathbf{Q}[X]/(g)$ as input and return the minimal polynomial f of some $\omega \in \mathcal{O}_\mathbf{K}$ so that $\deg f = d$ and so that f is simpler than g in some sense. The fields $\mathbf{K}$ and $\mathbf{Q}[X]/(f)$ are isomorphic.

We will focus on the reduction algorithm of [CD91] which is also described in [Coh93, Section 4.4.2]. The authors define the **size** of a monic degree d polynomial $f = \prod (X - \tau_j)$ as $\text{Size} f = \sum_{j=1}^{d} |\tau_j|^2$ where τ_j are the roots of f in $\mathbf{C}$ and this is the size that they minimize. We define

$$||f|| = (\sum_{j=1}^{d} |\tau_j|^2)^{\frac{1}{2}} = \text{Size} f^{\frac{1}{2}}.$$

This algorithm produces an f that is simpler in the sense that $||f||$ is upper bounded in terms of $|\Delta|$. In many cases $||f||$ is smaller than $||g||$.

This algorithm was implemented in `PARI/GP` as the function `polredabs` (short for "polynomial reduction absolute"). Since then additional improvements have been made in `PARI/GP`.

For f a defining polynomial of the field $||f||$ is related to $||\theta||$. We have

$$||\theta||^2 = \sum_{j=1}^{d} |\sigma_j(\theta)|^2 = \sum_{j=1}^{d} |\tau_j|^2 = ||f||^2.$$

The idea of the algorithm is to find a $\mathbf{Z}$-basis of $\mathcal{O}_\mathbf{K}$ and then compute a basis W of $\mathcal{O}_\mathbf{K}$ that is LLL-reduced with respect to the positive-definite quadratic form $|| \cdot ||^2$.

In the case $\mathbf{K}$ is primitive the output f of `polredabs`(g) will be the minimal polynomial of the first or second element of W in practice. The first element ω is often ± 1, so usually f is the minimal polynomial of the second element. Indeed, for some choices of δ this can even be proved, see [Bel04, Proposition 4.4]. We wish to know how large $||f||$ is. Applying Theorem 2.5 to ω we have

$$||\omega|| = ||f|| \leq c_\delta (d^{-1}|\Delta|)^{\frac{1}{2(d-1)}}$$

for primitive number fields.

In the case that the field is not primitive, `polredabs` will continue to walk the elements in W until an element is found that has degree d (i.e. does not live in a subfield). Specifically for quartic fields in practice we find the LLL reduced basis bound from Theorem 2.5 is too pessimistic because it gives an upper bound on $||f||$ of size $O(|\Delta|^{\frac{1}{2(d-i+1)}})$ for some i greater than 2 which is much larger than $|\Delta|^{\frac{1}{2(d-1)}}$. In practice we find $||f||$ is smaller than $c_\delta(d^{-1}|\Delta|)^{\frac{1}{2(d-1)}}$ for non-primitive fields as well as primitive fields.

The current implementation of `polredabs` returns a canonical defining polynomial for the field $\mathbf{K}$. This property is useful for some applications, for example, tabulations of the class group of all number fields of discriminant less than some bound. These canonical defining polynomials are used by the `LMFDB` as a canonical representation of number fields [LMF24a].

Effect of `polredabs` On Height

A notion of the largeness of a polynomial that the reader may think is more natural than Size is its height $H(f)$. As [GJ16] notes, despite [Coh93] motivating `polredabs` as a method for constructing defining polynomials of small height, `polredabs` is not well-suited to this (it *is* good at making $||f||$ small which is what we need in the sequel.). As `polredabs` is an important tool for us we want to make this aspect clear, so we will illustrate the negative impact of `polredabs` on the height with an example.

Example 2.12. *We consider*

$$g = X^4 - tX^3 - 6X^2 + tX + 1.$$

This g defines a simplest quartic field which we will discuss in Section 4.1. We will show there that these polynomials have a single large root and three small roots. For this example we let $t = 2^{40}$ and we define $f =$ `polredabs`(f). To see the effect of `polredabs` we place these polynomials side by side:

$$g = X^4 - 1099511627776X^3 - 6X^2 + 1099511627776X + 1$$
$$f = X^4 - 151115727451828646838274X^2 + 5708990770823839524233219435661706459854405632.$$

The negative impact of `polredabs` on the height is clear.

However, it should be emphasized that the application of `polredabs` gives a polynomial of better *size. That is,* $||f|| < ||g||$. *In particular, we have*

$$||g|| \approx 1.00 \cdot 2^{40} \textit{ and}$$
$$||f|| \approx 1.01 \cdot 2^{38}.$$

As the authors of [GJ16] note, the relationship between $||f||$ and $H(f)$ is a loose one, so we should not have an expectation that $H(f)$ will be small. Indeed, [Coh93] gives upper bounds for the coefficients of f. The largest, for the constant term, is $(||f||/d)^d$, and in practice the constant term of f is reasonably close to the upper bound.

Our experience with `polredabs` suggests two complementary heuristics about its effect on the height in practice for $||f|| \leq c|\Delta|^{1/(2(d-1))}$ for c a small constant. First, if there is a large difference in the absolute value of the roots of f then the output of `polredabs` has larger height than the original polynomial. Second, if all roots have similar absolute value then the height is essentially left unchanged. An extreme case of this is $f = X^d - t$. These polynomials are left unchanged by `polredabs` because the absolute values of the roots are the same ($|r| = t^{1/d}$).

The height of a polynomial is closely related to its discriminant so we might expect that disc(`polredabs`f) is much larger than disc f when `polredabs`f has larger height than disc f. Indeed, this is often the case. As f and `polredabs`f define isomorphic number fields the discriminants of both number fields are $|\Delta|$. Thus it is the index $[\mathcal{O}_{\mathbf{K}} : \mathbf{Z}[\theta]]$ that is increased.

Chapter 3

Prior Work

Contents

A framework for index-calculus class group algorithms emerged from the work of several authors in the 1980s including [PZ85], [HM89], and [Buc90] that all subsequent authors have adopted. The index-calculus framework has three major steps:

- First, fix a set of prime ideals of small norm that is large enough to generate $\mathrm{Cl}(\mathcal{O}_\mathbf{K})$.
- Second, generate relations between these prime ideals.
- Third, if possible, represent $\mathrm{Cl}(\mathcal{O}_\mathbf{K})$ and $\mathcal{U}(\mathcal{O}_\mathbf{K})$ using these relations by doing linear algebra on the system built from the relations. Otherwise, repeat the second step.

In Section 3.1 we describe the index-calculus framework for computing class groups and unit groups.

The focus of our thesis is on relation generation and so we give details on a number of relation generation approaches that are relevant to our goal in Sections 3.3–3.5. In Section 3.3 we consider ideal reduction relation generation which includes the approaches used by [Buc90] and [CDO97]. In Section 3.4 we give some details on the use of enumeration for relation generation which is an approach that is, unfortunately, not described in the literature, but appears to be the basis of the current state-of-the-art implementations.

Most relevant for the sequel are the subsections of Section 3.5 which discuss sieving relation generation. In this section we first describe Silverman's result on minimizing the size of the elements being tested for smoothness in quadratic sieving. This result is used in MPQS for integer factorization and for computing class groups. As discussed in the introduction, the self-initialization approach to relation generation developed by Jacobson [Jac99b], the self-initialization quadratic sieve SIQS, has been very successful but similar techniques have yet to be employed for fields of higher degree. In Section 3.5.2 we describe how multiple sieving polynomials are used in Jacobson's MPQS. In Section 3.5.3 we discuss how Jacobson's SIQS works and why it improves on MPQS.

3.1 Index Calculus Framework

In this section we give details on the index-calculus framework and discuss algorithm and implementation choices that can be made within it. As the index-calculus for computing class groups and related problems such as integer factorization is so well-studied many techniques have been developed over the years.

Recall that a factor base $\mathrm{FB}_{\mathbf{K}}$ is an $|\,\mathrm{FB}_{\mathbf{K}}\,|$-tuple of prime ideals $\mathfrak{p}$ with $N(\mathfrak{p}) < B$ with $B \in \mathbf{Z}_{>0}$. By $\mathfrak{a} = \mathrm{FB}_{\mathbf{K}}^{\mathbf{v}}$ we mean that $\mathfrak{a}$ is B-smooth and the integer vector $\mathbf{v} = (v_i)_{i \leq |\,\mathrm{FB}_{\mathbf{K}}\,|}$ satisfies $\mathfrak{a} = \prod \mathfrak{p}_{\mathbf{i}}{}^{v_i}$.

Now let

$$\phi : \mathbf{Z}^{|\,\mathrm{FB}_{\mathbf{K}}\,|} \to \mathcal{I}/\mathcal{P} \qquad \phi(\mathbf{v}) = \prod_{i=1}^{|\,\mathrm{FB}_{\mathbf{K}}\,|} [\mathfrak{p}_i]^{v_i}.$$

The map ϕ is a homomorphism and the kernel of ϕ is a sublattice of $\mathbf{Z}^{|\,\mathrm{FB}_{\mathbf{K}}\,|}$. It is possible to choose $\mathrm{FB}_{\mathbf{K}}$ so that it contains a set of generators of $\mathrm{Cl}(\mathcal{O}_{\mathbf{K}})$ and in this case, the homomorphism ϕ is surjective. From this we know

$$\mathbf{Z}^{|\,\mathrm{FB}_{\mathbf{K}}\,|} / \ker(\phi) = \mathrm{Cl}(\mathcal{O}_{\mathbf{K}}).$$

To find $\mathrm{Cl}(\mathcal{O}_{\mathbf{K}})$ all we need to do is to find a basis of $\ker(\phi)$.

Elements $\mathbf{v}$ of $\ker(\phi)$ satisfy $\mathrm{FB}_{\mathbf{K}}^{v} \sim (1)$ and so $\ker(\phi)$ can be used to simultaneously work towards our other goal of finding the logarithmic embedding of a system of fundamental units, which we represent as the matrix $\mathrm{V}(\mathcal{O}_{\mathbf{K}})$. We do this by working with **relations** which are pairs of the form $(\mathbf{v}, \operatorname{Log}\gamma)$ such that $\mathrm{FB}_{\mathbf{K}}^{\mathbf{v}} = (\gamma)$. We let Λ be the set of relations.

Then Λ is a lattice as for any relations $(\mathbf{v}_1, \gamma_1), (\mathbf{v}_2, \gamma_2)$ and $a \in \mathbf{Z}$ we have $(\mathbf{v}_1, \operatorname{Log}\gamma_1) + (\mathbf{v}_2, \operatorname{Log}\gamma_2)$ and $a(\mathbf{v}_1, \operatorname{Log}\gamma_1)$ are also relations. In [Buc90] it is shown that the determinant of Λ is hR if $\mathrm{FB}_{\mathbf{K}}$ generates the entire $\mathrm{Cl}(\mathcal{O}_{\mathbf{K}})$.

To find Λ we generate a set of relations and work with the sublattice Λ' that they generate. From this sublattice we can obtain integer multiples $\widetilde{h}$ of h and $\widetilde{R}$ of R. If $\widetilde{h} = h$ and $\widetilde{R} = R$. then Λ' is equal to the full lattice Λ.

Our discussion will mostly be about an approach that works only with $\operatorname{Log}\gamma$ and does not need γ. For this approach we can even discard γ after $\operatorname{Log}\gamma$ is computed. We also will touch on so-called "saturation" methods. These methods work with γ as well as $\operatorname{Log}\gamma$.

To construct a relation we start with a B-smooth **initial ideal** $\mathfrak{a}$ and generate $(\mathfrak{b}, \gamma)$ so that

$$\mathfrak{a}\mathfrak{b} = (\gamma).$$

If $\mathfrak{b}$ is also B-smooth then there is some vector $\mathbf{v}$ so that $\mathfrak{a}\mathfrak{b} = \mathrm{FB}_{\mathbf{K}}^{\mathbf{v}} = (\gamma)$ and $(\mathbf{v}, \gamma)$ is a relation.

A basis of $\ker(\phi)$ is found by constructing an integer matrix M with rows that generate a sublattice of $\ker(\phi)$. The matrix has $|\,\mathrm{FB}_{\mathbf{K}}\,|$ columns and the ith row in M is $\mathbf{v}_i$ from the ith relation $(\mathbf{v}_i, \gamma_i)$. The HNF has the form $\mathrm{HNF}(M) = [0|H]$ for a square matrix H and we compute $\widetilde{h} = \det(H)$ which, if H is full-rank and $\mathrm{FB}_{\mathbf{K}}$ generates $\mathrm{Cl}(\mathcal{O}_{\mathbf{K}})$, will be a nonzero multiple of h.

A basis for the lattice of images of the units under Log is found using the kernel of M. This kernel is computed simultaneously with the HNF and we store the result in a matrix A. We construct a matrix $M_{\log}$ of the $\operatorname{Log}\gamma$. The matrix has $r_1 + r_2 - 1$ columns and the ith row in $M_{\log}$ is $\operatorname{Log}\gamma_i$ where γ_i is taken from the ith relation $(\mathbf{v}_i, \gamma_i)$. We compute the product of $M_{\log}$ and A and store it in a matrix C. The integer multiple $\widetilde{R}$ of R is the determinant of a submatrix V' of C representing a basis.

We can efficiently check if $\Lambda = \Lambda'$ using a result of [Bac95] that is conditional on GRH. When we have expanded Λ' sufficiently that $\Lambda = \Lambda'$, which is a condition we can check, assuming GRH, using a result of [Bac95], we know that V' represents a system of fundamental units under the logarithmic embedding and so V' is the matrix the algorithm should return for $\mathrm{V}(\mathcal{O}_{\mathbf{K}})$. We compute the class group using the Smith Normal Form S of the l-dimensional submatrix of H formed by omitting columns and rows for which the corresponding diagonal element is 1. The class group is $\mathrm{Cl}(\mathcal{O}_{\mathbf{K}}) = \mathbf{Z}/s_1\mathbf{Z} \times \cdots \times \mathbf{Z}/s_l\mathbf{Z}$ for $l \in \mathbf{Z}_{\geq 1}$ and $s_i \in \mathbf{Z}_{>1}$ where the s_i are the diagonal entries of S. The algorithm returns $(s_i)_{i \leq l}$ for $\mathrm{Cl}(\mathcal{O}_{\mathbf{K}})$. Finally, if it is not known that the factor base we used generates the entire class group assuming GRH, we verify this. As a result, the output is correct assuming the validity of GRH.

The remainder of this section describes the steps in the framework in more detail.

3.1.1 Initialization

Factor Base

The first step of the framework that we have not already covered is choosing the bound B used to define the factor base. What has been found by many authors of class group implementations is that for the range of $|\Delta|$ for which the class group can be computed, the optimal B is notably smaller than what is suggested by a time complexity analysis. The optimal smoothness bound depends on the implementation and so B must be determined experimentally. B has a large impact on the time required for both relation generation and linear algebra.

We start with its impact on the linear algebra stage because the effect of B is more straightforward in this case. The size of the factor base $|\mathrm{FB}_{\mathbf{K}}|$ is a strictly-increasing function of B. The number of relations we must generate to complete the algorithm is slightly more than $|\mathrm{FB}_{\mathbf{K}}|$. The time for the linear algebra stage is dominated by the algorithms that are executed on the $O(|\,\mathrm{FB}_{\mathbf{K}}\,| \times |\,\mathrm{FB}_{\mathbf{K}}\,|)$ matrix M, such as the HNF. Thus the time for linear algebra is a strictly-increasing function of B.

Regarding the relation generation stage, the smoothness bound B affects the efficiency in two ways: first, it determines the size of $\mathrm{FB}_{\mathbf{K}}$ and hence the number of relations that must be generated in order to complete this stage. Second, it controls the probability for elements to be B-smooth and thus the expected number of relations we generate in a fixed time interval. Consequently, the relationship between B and the total time for the relation generation stage is more complex than for the linear algebra stage.

Due to its large impact on both relation generation and linear algebra, the choice of B is an important factor in the speed of the overall algorithm.

In [Buc90] the goal is to prove the complexity of the algorithms and the B that minimizes the complexity of the overall algorithm is subexponential in $\log(|\Delta|)$ (see [JW09, Section 13.7.1] and Step 9 of [Coh93, Algorithm 6.5.7]). In practice, it has been observed that this B is much larger than needed.

For a particular implementation the optimal choice of B is made empirically based on constraints arising from the linear algebra stage (i.e. the maximum size of M that can be handled in memory). Recall that the bounds discussed in Section 2.6 give an integer $T_{\min}$ so that the prime ideals that are smaller than $T_{\min}$ generate the class group under GRH. For a particular implementation the optimal choice of B could be larger or smaller than the bounds discussed in that section.

Approximating hR

We use an approximation $\widetilde{hR}$ of hR to determine if $\Lambda = \Lambda'$. This approximation is obtained from the analytic class number formula which we discussed in Section 2.6. We cannot compute $\lim_{s\to 1}(s-1)\zeta_{\mathbf{K}}(s)$ to an arbitrary precision efficiently. What is possible is to efficiently compute an approximation $\widetilde{hR}$ so that

$$\widetilde{hR} \leq hR \leq 2\widetilde{hR}$$

under GRH.

The approach used by [Coh93, Section 6.5.2] is to compute the Euler product for the prime ideals of norm less than or equal to $12(\log|\Delta|)^2$. We let,

$$z = \prod_{p \le B} \frac{1 - \frac{1}{p}}{\prod_{N(\mathfrak{p}) \le B}\left(1 - \frac{1}{N(\mathfrak{p})}\right)}$$

and define $\widetilde{hR} = \frac{m\sqrt{|\Delta|}}{2^{r_1}(2\pi)^{r_2}} z$. It is shown that $\widetilde{hR} \le hR \le 2\widetilde{hR}$ in [CDO97] using a result of [Bac95, Section 8].

3.1.2 Relation Generation

To construct relations we start with a B-smooth initial ideal $\mathfrak{a} = \prod_{i \le |\,\mathrm{FB}_{\mathbf{K}}\,|} \mathfrak{p}_{\mathbf{i}}{}^{w_i} = \mathrm{FB}_{\mathbf{K}}^{\mathbf{w}}$ and generate $(\mathfrak{b}, \gamma)$ so that $\mathfrak{a}\mathfrak{b} = (\gamma)$ and $N(\mathfrak{b})$ is B-smooth. Many different methods for generating relations are used. As this thesis is focused on improving relation generation, Section 3.1.2 is devoted to describing these. In particular, that section focuses on methods that construct the pair $(\mathfrak{b}, \gamma)$ of B-smooth norm. In this section we will say what is done after we generated them.

At this point $\mathfrak{b}$ is likely to be B-smooth but this is not yet verified. We use the prime factors of $N(\mathfrak{b})$ to verify this. Recall from Chapter 3 that we can efficiently compute the valuation z_i of $\mathfrak{b}$ at $\mathfrak{p}_{\mathbf{i}}$ using the algorithm from [Bel04, Section 5.3]. The B-smoothness of $\mathfrak{b}$ is tested by computing the valuation z_i at every $\mathfrak{p}_{\mathbf{i}}$ in $\mathrm{FB}_{\mathbf{K}}$ and checking if $\mathfrak{b} = \prod \mathfrak{p}_{\mathbf{i}}{}^{z_i}$. In the case $\mathfrak{b}$ is B-smooth $(\mathbf{w} + \mathbf{z}, \gamma)$ is a relation where $\mathbf{w} = (w)_{i \le |\,\mathrm{FB}_{\mathbf{K}}\,|}$. Each time a relation is found the row vector $\mathbf{w} + \mathbf{z}$ is added to M and the row vector $\mathrm{Log}(\gamma)$ is added to $M_{\log}$.

For a fixed smoothness bound B the probability that $\mathfrak{b}$ is B-smooth depends on the size of $N(\mathfrak{b})$ (see [BF14]). A larger $N(\mathfrak{b})$ means $\mathfrak{b}$ is less likely to be B-smooth so relation generation methods must ensure $N(\mathfrak{b})$ is kept small.

Following Hafner-McCurley [HM89] many authors do relation generation in two stages. First, $O(|\,\mathrm{FB}_{\mathbf{K}}\,|)$ **initial** relations are generated then we iteratively do some linear algebra to determine if these relations are sufficient to get the class number and regulator. If not we add a small number of additional relations. Both initial and additional relations could be generated in a random manner or in a non-random manner. In the latter case we call them **targeted** relations. One important factor in the efficiency of the framework is the number of random relations k_{req} we generate before any linear algebra is done. We require Λ to be a full-rank sublattice of Λ'. To get Λ' to have full rank, at least $|\,\mathrm{FB}_{\mathbf{K}}\,|$ relations are needed but in practice we need more than this. We describe how authors have choosen the number of initial relations to generate in Section 3.1.7.

Number of Initial Relations

The goal of relation generation is to obtain enough full relations so that

$$\widetilde{hR} < \widetilde{h}\widetilde{R} < 2 \cdot \widetilde{hR}. \tag{3.1}$$

Note that $\widetilde{h}$ is defined as 0 if the module generated by the vectors $\mathbf{v}$ in our relations $(\mathbf{v}, \operatorname{Log} \alpha)$ does not have full rank. Thus we must find at least $|\operatorname{FB}_{\mathbf{K}}|$ relations for equation (3.1) to be satisfied.

A standard approach has been to economize on the number of relations passed to the HNF as much as possible. In this approach we stop generating full relations after we have $|\operatorname{FB}_{\mathbf{K}}| + c$ of them for a small $c > 0$. After generating the initial $|\operatorname{FB}_{\mathbf{K}}| + c$ relations, we iteratively check the rank of M and add a small number of additional relations until we reach full rank.

Historically, implementations of index-calculus class group algorithms computed the HNF without a structured Gaussian elimination. In this context, two reasons for economizing on the number of relations passed to the HNF are:

- the time to compute the HNF on a matrix with a fixed number of columns grows rapidly with the number of rows,
- implementations of HNF algorithms are often optimized for the case where the input matrix is close to being square.

This explains why algorithms that do not perform a structured Gaussian elimination want c to be small. On the other hand $c = 0$ will almost never be enough relations. The author of an algorithm can choose c so that $|\operatorname{FB}_{\mathbf{K}}| + c$ has the minimum value they think will be necessary for all $|\Delta|$ sizes they are interested in. Optimizing c this way can give a small performance improvement over $c = 0$, particularly for small $|\Delta|$.

For reference, we mention some values of c that have been used in [Jac99b], [BJN+99], and [BJ10]:

- [Coh93, Algorithm 6.5.7] uses $c = 10$
- [Jac99b] uses c between 5 and 10
- [BJN+99] uses $c = 20$

- [BJ10] uses $c = 100$

The use of $c = 100$ in [BJ10] applies only to when the large prime variation is not used. When the large prime variation is used [BJ10] does not give a value for c but only requires more than $|\,\mathrm{FB}_{\mathbf{K}}\,|$ relations are found. For class groups of higher degree number fields [Coh93, Algorithm 6.5.7] mentions that the choice $c = 10$ is only used for consistency with the quadratic class group algorithm described in an earlier chapter.

Choosing the number of relations to be of the form $|\,\mathrm{FB}_{\mathbf{K}}\,| + c$ for a small c is a common practice but forcing the number of initial relations to be so small is not necessarily optimal. Structured Gaussian elimination is now the standard practice for index-calculus class group algorithms. As observed in [BKV19], if structured Gaussian elimation is being done, having a larger number of initial relations can be beneficial:

> "The goal of [relation generation] is to generate sufficiently many relations such that the subsequent steps are able to determine the class group. In practice, this usually means that we can stop collecting relations when the number of relations slightly exceeds the number of prime ideals contained in their decompositions... However, a bigger excess often reduces the running time of the subsequent steps significantly."

The "number of prime ideals contained in their decompositions" means $|\,\mathrm{FB}_{\mathbf{K}}\,|$ in our notation. One reason that more initial relations is helpful is that the rank of M will be closer to being full. Another reason is that in a larger set of relations we may have relations that have smaller entries at certain columns allowing structured Gaussian elimination to produce a better result (lower density and coefficient size).

Once structured Gaussian elimination has been completed, the excess rows can be discarded so that the matrix is slightly overdetermined. Thus choosing a number of relations of the form $a|\,\mathrm{FB}_{\mathbf{K}}\,|$ for $a > 1$ can be helpful. This is the approach used by [BKV19].

3.1.3 Gaussian Elimination Techniques

After generating relations we have a sparse matrix M. One way to compute properties of sparse integer matrices such as their HNF, Smith Normal Form, or nullspace is to first use Gaussian elimination to reduce the dimensions of the matrix. Next, the algorithm for computing the property is then applied to this smaller matrix.

Some definitions: The **weight of a row** is the number of nonzero entries in the row. The **weight of a matrix** is the number of nonzero entries in the matrix. The **density of a row** is its weight divided by its length. The **density of a matrix** is its average row density.

Naive Gaussian elimination can be used to reduce the dimensions of M but it leads to a rapid increase in the density and size of coefficients. The goal of **structured Gaussian elimination** is to obtain a matrix of smaller dimensions while minimizing the increase in density and size of coefficients using the standard row operations as well as

- the deletion of rows (as relations can be deleted)
- the deletion of zero columns that correspond to prime ideals that are not in $\mathrm{FB}_{\mathbf{K}}$

Note that some authors such as [Cav02] use the term structured Gaussian elimination more narrowly.

A number of different elimination strategies are available. Jacobson uses an elimination strategy of [HM97] that had been implemented in the library `LiDiA` (now defunct). The strategy used in `Magma` is Markowitz pivoting (see [CBFS23a] and its reference [DER86, Section 9.2]). Another approach was introduced by Cavallar [Cav02] which is now standard for Number Field Sieve implementations.

As Cavallar's elimination strategy was defined in the context of integer factorization where linear algebra is done modulo 2 it is not obvious that it should be useful for computing class groups. Biasse in [Bia10] was the first to apply this approach to computing the class group of imaginary quadratic fields. He shows that it is effective.

3.1.4 Large Prime Variations

Many index-calculus algorithms use an optimization called a large prime variation.

The simplest is the single large prime variation. This approach has been used in a number of index-calculus implementations, for example [Jac99b]. In this approach we fix a large prime bound LB. By checking the factorization of the norm of γ_1 we can determine if $(\gamma_1) = \mathrm{FB}_{\mathbf{K}}^{\mathbf{v}} \cdot \mathfrak{p}$ for $\mathfrak{p}$ a prime which is outside of $\mathrm{FB}_{\mathbf{K}}$ satisfying $N(\mathfrak{p}) \leq LB$. We call the triple

$$(\mathbf{v_1}, \operatorname{Log}\gamma_1, \mathfrak{p}_1)$$

a **partial relation**. In the case that we find a prime $B \leq p \leq LB$ dividing the norm we construct a partial relation. We identify the $\mathfrak{p}$ dividing (γ_1) by finding all prime ideals above p and testing which one divides (γ_1).

Suppose for a prime $\mathfrak{p}$ we have l partial relations $(\mathbf{v_1}, \operatorname{Log}\gamma_1, \mathfrak{p}), \ldots, (\mathbf{v_l}, \operatorname{Log}\gamma_l, \mathfrak{p})$. There are $\binom{l}{2} = l \cdot (l-1)/2$ possible quotients (γ_i/γ_j) for $i \neq j$ each of which factors over the $\mathrm{FB}_{\mathbf{K}}$ and so gives us a full relation.

Biasse and Jacobson in [BJ10] describe the traditional approach to combining partial relations into a full relation this way: "In the case of 1-partial relations, any pair of relations involving the same large prime p were recombined into a full relation." This approach was implemented by Jacobson in [Jac99b] and compared with the same algorithm without the large prime variation. The version with the large prime variation was found to be slower.

Based on Cavallar's ideas in [Cav02] there appear to be three requirements that a strategy for combining partial relations into full relations should meet:

- the set of full relations spans all the full relations that are possible,
- the set of new full relations do not have linear dependencies that could easily be avoided,
- the increase in density and increase in the size of coefficients of M is minimized.

Using structured Gaussian elimination full relations can be constructed by making all the column entries 0 in the columns corresponding to the large primes. Indeed, Cavallar [Cav02] observes that combining partial relations into full relations and doing structured Gaussian elimination can be done in a single step. Number Field Sieve implementations such as `CADO-NFS` follow this approach (the whole step is usually called "filtering").

We briefly mention the difference in the number of relations that are needed when a large prime variation is used and when it is not and how this affects how we choose parameters. When a large prime variation is used more relations are needed. From previous work on large prime variations such as [BJ10],[BKV19] it is well-known that the number of partial relations that minimizes the overall running time of the algorithm will be considerably larger than $|\,\mathrm{FB}_{\mathbf{K}}\,|$. This observation holds true across many implementations. Of course, the number of partial relations that should be generated depends heavily on the implementation and must be determined experimentally. As an illustration, the authors of [BKV19] computed the class group of a 512 bit discriminant using a large prime variation (with three primes). They set $|\,\mathrm{FB}_{\mathbf{K}}\,|$ to be 7 million. In order to compute the class group efficiently they computed 319.5 million relations (the large prime bound was approximately

30 million). Thus they used a number of relations that was approximately 45.6 times larger than $|\,\mathrm{FB}_{\mathbf{K}}\,|$.

3.1.5 Linear Algebra

Many possible approaches exist for using linear algebra to deduce the class group and unit group from a set of relations. This thesis is primarily about improving the relation generation stage so an exhaustive survey of all these approaches is beyond the scope of this thesis. In order to implement an efficient algorithm we tried several approaches (described in Section 5.2.4). In this section we discuss prior work related to the approaches that we tried.

The first computation we do with the $k_{\mathrm{req}} \times |\,\mathrm{FB}_{\mathbf{K}}\,|$ integer matrix M is checking if M has full-rank. This can be done using standard Gaussian elimination modulo a suitable prime (see [Coh93, Chapter 2]) without computing the HNF of M directly. If M does not have full-rank we add more relations to Λ' as described below.

In Section 3.1.5 the approach to the linear algebra which uses the Hermite normal form is discussed. The main ideas of this approach seem to originate with the work of Hafner and McCurley [HM89] on computing class groups of quadratic number fields. This approach is also used, for example, in [Jac99b] and [Coh93]. We then discuss variations of it that have been used.

Computing the Class Group Using the HNF and its Transformation Matrix

We now consider computing the class group by computing the HNF and its transformation matrix.

Once M has full-rank we compute the HNF of M directly. Computing the HNF and transformation matrix for an integer matrix with $|\,\mathrm{FB}_{\mathbf{K}}\,|$ columns and $O(|\,\mathrm{FB}_{\mathbf{K}}\,|)$ rows is time-consuming. The nullspace A of M can be computed simultaneously with $\mathrm{HNF}(M) = [0|H]$ (see [Coh93, Chapter 4]). The state-of-the-art algorithm in practice appears to be due to [MW01a] which was implemented and improved by Allan Steel in `Magma` (a description has not been published). Improvements to [MW01a] are described in [PS10] which is the basis for the implementation in `Flint`.

We compute $\widetilde{h} = \det(H)$. Under GRH $\mathrm{Cl}(\mathcal{O}_{\mathbf{K}})$ is generated by the prime ideals in $\mathrm{FB}_{\mathbf{K}}$ and so we have $h \mid \widetilde{h}$.

Computing the Unit Group

The matrix $C = AM_{\log}$ has rows $\operatorname{Log}(u)$ for $u = \prod \gamma_i^{v_i}$ for integer v_i where u is a unit of $\mathbf{K}$ and this set of units is a subgroup of $\mathcal{U}(\mathcal{O}_{\mathbf{K}})$. As we have taken $k_{\text{req}} > |\operatorname{FB}_{\mathbf{K}}| \gg d$ it is extremely unlikely that this group of units is not full-rank. If the subgroup is full-rank the rows of C generate a sublattice of the lattice generated by a system of fundamental units under Log.

Given nonnegative real numbers a, b which are approximate integer multiples of a positive real number x we define the **rgcd** (short for "real gcd") as the largest real number d that is an approximate integer multiple of x and divides a and b. The rgcd can be computed efficiently for $r > 0.2$ using [Coh93, Algorithm 5.9.3].

By performing row operations on the matrix C with the RGCD algorithm we compute a basis of this sublattice as a matrix V'. This can be done using [Coh93, Algorithm 6.5.7] using the embedding Log instead of the complex logarithmic embedding used by [Coh93]. In this algorithm, the determinant $\widetilde{R}$ of V' is computed simultaneously.

Unlike all the previous steps in the algorithm, which do not depend on the rank r of $\mathbf{K}$, the cost of this step depends on r as C has r columns. However, as r is much smaller than $|\operatorname{FB}_{\mathbf{K}}|$ this step is not a computational bottleneck.

Additional Relations

We now turn to verifying that we have the full lattice of relations and adding additional relations if we do not. Using the truncated Euler product in Section 3.1 we have computed an approximation $\widetilde{hR}$ that satisfies $\widetilde{hR} < hR < 2 \cdot \widetilde{hR}$ under the assumption of GRH. No integer multiples of hR other than hR are in the interval $[\widetilde{hR}, 2 \cdot \widetilde{hR}]$ so if $\widetilde{h}\widetilde{R} < 2\widetilde{hR}$ then we must have $\widetilde{h} = h$, $\widetilde{R} = R$, and $\Lambda' = \Lambda$. Thus we know the $\operatorname{Cl}(\mathcal{O}_{\mathbf{K}})$ and $\operatorname{V}(\mathcal{O}_{\mathbf{K}})$ found using the relations in Λ' are correct once it has been verified that $\operatorname{FB}_{\mathbf{K}}$ generates $\operatorname{Cl}(\mathcal{O}_{\mathbf{K}})$.

If either Λ' does not have full rank (which, given our choice of k_{req}, will correspond to $\widetilde{h} = 0$ with extremely high probability), or we have $\widetilde{h}\widetilde{R} \geq 2\widetilde{hR}$ we iteratively do the following:

1. generate a small number of new relations
2. update M and $M_{\log}$,
3. update $(\widetilde{h}, \widetilde{R})$

4. check if $\widetilde{h}\widetilde{R} \geq 2\widetilde{hR}$.

One approach is to generate a small number of random relations. Alternatively, we can target relation generation by trying to get a relation $(\mathbf{v}, \gamma)$ so that (γ) is divisible by a specific prime in $\mathrm{FB}_{\mathbf{K}}$. Generating these relations is discussed in Section 3.1.7.

3.1.6 Linear Algebra Variations

The approach of computing the HNF and its transformation matrix is well understood. This is due to the fact that the HNF lets us express a set of generators for the class group in terms of power-products of small elements. However, computing the HNF and its transformation matrix is not essential to computing class groups and unit groups. Approaches of this kind have been developed for real quadratic number fields. As was observed by Jacobson in [Jac99b, Section 4.5], computing the transformation is expensive and sometimes unnecessary. Not computing the transformation reduces the number of intermediate matrix entries that are manipulated in each step of the HNF. This is extremely valuable due to the problem of coefficient explosion.

Turning to the computation of the unit group, we can use methods based on solving a systems of linear equations over the integers. The advantage of doing this is that solving a systems of linear equations over the integers can be done very quickly in practice. For real quadratic number fields one method form introduced by Vollmer [Vol00] constructs elements of the nullspace of M. Once we have found $|\mathrm{FB}_{\mathbf{K}}|$ relations so that M has full rank we find m additional random relations $\{(\mathbf{r_i}, \operatorname{Log}\alpha_i)\}_{i=1}^{m}$ so that r_i does not appear in M. Then solve the system of equations $\mathbf{X}M = \mathbf{r_i}$ in the integers for a vector $\mathbf{x_i}$. As the unit rank is 1 we have that $\mathbf{x_i}M_{\log}$ is a multiple of the regulator. We let R_i be the real number so that $R_i = \operatorname{Log}(|\alpha_i|) - \mathbf{x_i}M_{\log}$. After enough random $\mathbf{x_i}$ are constructed we will find a pair $\mathbf{x_1}$ and $\mathbf{x_2}$ so that R_1 and R_2 have rgcd equal to the regulator.

Its potential to be more practical than the first approach is based on two observations:

- Computing the HNF without the transformation matrix is an order of magnitude faster than computing the HNF with the transformation matrix.
- Solving systems of linear equations in the integers is faster than computing the HNF with the transformation matrix.

Using this approach for computation of the regulator of a real quadratic field was studied in [BJS10]. They found it to be significantly faster than the HNF approach.

Another approach to computing the regulator without computing the HNF is to compute an integer basis A for the sublattice of all integer vectors in the kernel of M. Algorithms for this problem exist in the software `Magma` and the Integer Matrix Library IML which are faster than solving this problem by computing the HNF transformation matrix. We then proceed in the same manner as above.

3.1.7 Additional Relations & Hybrid Strategies

After the initial relation generation we often have a lattice of relations Λ' that does not quite have full rank. If the matrix is almost full rank, e.g. the rank is at least 97% of $|\,\mathrm{FB}_{\mathbf{K}}\,|$, we can try to generate relations that are divisible by specific prime ideals. The linearly dependent columns of an integer matrix can be computed in a number of different ways. The simplest practical method may be to work modulo a random wordsize prime and use Gaussian elimination to find the non-pivot columns. Due to the size of the modulus, this approach will only rarely find an extra pivot column that is not a pivot of the integer matrix. The column indices identify certain primes $\mathfrak{p}$ in the factor base.

Selecting such a prime $\mathfrak{p}$ we can target relation generation by trying to generate a relation $(\mathbf{v}, \gamma)$ so that (γ) is divisible by it. Any of the relation generation methods we will subsequently discuss in this chapter can be used for targeting. Targeting can be be done with an ideal reduction approach (see [Jac99b, Ch 3.1.4]). In Jacobson's [Jac99b] implementation (see [Jac99b, Algorithm 4.3]) targeting is done using sieving relation generation after SIQS relation generation has been used to generate random relations. In [BF12] targeting is done with the enumeration approach in `PARI/GP`.

After finding a small number of relations for each such $\mathfrak{p}$ we recompute the pivot columns to see if the matrix is full rank.

After targeting relations to reach full rank we may not be finished since we still may only have a sublattice of the lattice of relations. We can expand this sublattice by finding additional random relations. Any of the relation generation methods in this chapter can be used for doing this.

The strategies for finding additional relations in [Jac99a] and [BF12] suggest that the best algorithms are often hybrids of more than one approach to relation generation. State-of-the-art implementations for the Number Field Sieve also use a hybrid approach. Describing these implementations Thomé [Tho22, Lec 10] explains that

"There is no single way to do relation collection. In practice, the most efficient way involves a blend of: Sieving (in order to detect); Trial-division; Re-sieving (in order to factor); ECM; [and] Product trees"

3.1.8 Verification

If B is chosen smaller than $T_{\min}$ then at some point we must verify that the class group is generated by the prime ideals in the factor base.

In this verification we check that each prime of norm up to $T_{\min}$ is contained in the subgroup of $\mathrm{Cl}(\mathcal{O}_{\mathbf{K}})$ generated by $\mathrm{FB}_{\mathbf{K}}$. This is done by finding for each prime ideal $\mathfrak{p}$ of norm in the interval $[B, T_{\min}]$ a factorization of the form $\mathfrak{p} = (\gamma)\,\mathrm{FB}_{\mathbf{K}}^{\mathbf{v}}$.

The additional verification step makes the design of an efficient index-calculus algorithm more challenging. Normally such an algorithm has two time consuming stages: relation generation and linear algebra. Verification adds a third time consuming step that needs to be considered. Tuning the implementation parameters so that the total running time is minimal is already a challenge. However, it may be worthwhile for applications where the verification is only necessary for some number fields or when the verification can be deferred until a later time.

Jacobson in [Jac99b] found the verification step was worthwhile. More precisely, he found that the time taken by the relation generation and linear algebra steps in his SIQS implementation could be reduced by setting B smaller than the Bach bound for quadratic fields $6(\log|\Delta|)^2$ (the best bound at the time).

3.2 Work on Related Problems

Before proceeding to our discussion of index-calculus relation generation we briefly mention some of the most important results related to computing class groups with a different focus from ours.

3.2.1 Leveraging Subfields

The focus of this thesis is on computing class groups and unit groups for number fields without special properties such as subfields. However, if the number field has subfields then in some cases it is possible to reduce the problem to the computation of the class group and unit group of the subfields.

Given the class groups of the three quadratic subfields of a biquadratic number field a theorem of Kuroda [Kur50] allows the odd part of the class group of $\mathcal{O}_{\mathbf{K}}$ to be computed. A similar result can be used to compute the unit group of $\mathcal{O}_{\mathbf{K}}$. Using these results, algorithms for the class group and unit group of multiquadratic number fields, which have the form $\mathbf{Q}(\sqrt{a_1}, \dots, \sqrt{a_k})$ with a_i squarefree positive integers that are multiplicatively independent modulo squares, are given in [BBdV+17] and [BvV19] whose efficiency is a function of $\prod a_i$.

These algorithms are efficient in practice and we believe this approach is a better choice than index-calculus algorithms for biquadratic fields. Generalizations of these algorithm have recently been given. At present, these are less relevant to the practical computation of class groups and unit groups of quartic number fields.

3.2.2 Time Complexity

Our focus is on computing class groups and unit groups in practice. The study of the time complexity of algorithms has often informed practical computation. Unfortunately, a discussion of the time complexity of our results in the sequel is beyond the scope of this thesis but we mention some of the results.

The time complexity is written using the function

$$L_{|\Delta|}[a, b] = e^{(b+o(1))(\log|\Delta|)^a (\log\log|\Delta|)^{1-a}}$$

for constants $0 \leq a \leq 1$ and $b \geq 0$ and authors that have not determined the constant b write $L_{|\Delta|}[a]$.

The state-of-the-art time complexity for number fields of arbitrary degree is $L_{|\Delta|}[\frac{1}{2}, 1.7]$ for the algorithm of [Buc90]. For quadratic number fields algorithms with better time complexity exist. The Hafner-McCurley algorithm [HM89] has heuristic time complexity $L[\frac{1}{2}, \sqrt{2}]$. The second term in the L notation can be slightly improved (see [JW09, Chapter 13]).

In the case of families of number fields for which the degree d grows as a function of $\log(|\Delta|)$ the algorithm of [BF14] has time complexity $L_{|\Delta|}[\frac{2}{3} + \epsilon]$ for ϵ arbitrarily small. In [Gél17] this algorithm is analyzed in detail, including deriving the first and second terms in the L-notation.

All of these time complexities are derived under heuristic assumptions. All of them require GRH but a variety of other heuristics are used.

3.3 Ideal Reduction Relation Generation

Shortest Vector Ideal Reduction Relation Generation

An ideal reduction algorithm is given by Buchmann [Buc90, Section 3.2] that is valid for number fields of arbitrary degree. In this paper the primary goal is to prove a heuristic subexponential time complexity rather than to achieve efficiency in practice. We call his approach shortest vector ideal reduction to distinguish it from the LLL ideal reduction we discuss later in this section. This approach finds an element γ corresponding to a shortest vector in $\sigma(\mathfrak{a})$. As we want the output ideals $\mathfrak{b}$ to be chosen in a random manner, the v-norm $||\cdot||_{\mathbf{v}}$ is used.

For the method of [Buc90] we choose random nonnegative integers $w_i \leq e$ for some positive integer e to define the vector $\mathbf{w} = (w_i)_{i \leq |\,\mathrm{FB}_{\mathbf{K}}\,|}$, choose random nonnegative integers $v_i \leq e$ for $i \leq r_1 + r_2$, and set $v_{i+r_2} = v_i$ for $r_1 + r_2 < i \leq d$ to define the vector $\mathbf{v} = (v_i)_{i \leq d}$. We then compute the ideal $\mathfrak{a} = \prod_{i \leq |\,\mathrm{FB}_{\mathbf{K}}\,|} \mathfrak{p}_{\mathbf{i}}{}^{w_i} = \mathrm{FB}_{\mathbf{K}}^{\mathbf{w}}$. Using an enumeration algorithm we compute the shortest vector in $\sigma(\mathfrak{a})$ using the norm $||\cdot||_{\mathbf{v}}$. We find the element $\gamma \in \mathbf{K}$ corresponding to this vector and define $\mathfrak{b}$ so that $\mathfrak{a}\mathfrak{b} = (\gamma)$. For the proof of the algorithm's subexponential time complexity we need a large e. Buchmann uses $e = |\Delta|$.

Recall we want $\mathfrak{b}$ to have small norm. As a consequence of the bound on a shortest vector $\mathbf{v}_1$ from Chapter 3 we have

$$N(\mathfrak{b}) \leq \mu_d^{d/2} \cdot |\Delta|^{\frac{1}{2}}.$$

The time required to find the pair $(\mathfrak{b}, \gamma)$ is dominated by the time to compute the shortest vector in $\sigma(\mathfrak{a})$ but this problem can be solved efficiently when d is small.

LLL Ideal Reduction Relation Generation

In [Coh93] ideal reduction is implemented using LLL lattice basis reduction rather than by computing the shortest vector in $\sigma(\mathfrak{a})$.

In this method we construct $\mathbf{w}$ similarly to Section 3.3. A difference is that due the focus on obtaining a practical algorithm [Coh93] suggests using a much smaller e of 20. This makes the cost of the ideal arithmetic much smaller. We choose random nonnegative integers $w_i \leq e$ for some positive integer e to define the vector $\mathbf{w} = (w_i)_{i \leq |\,\mathrm{FB}_{\mathbf{K}}\,|}$, choose random nonnegative integers $v_i \leq e$ for $i \leq r_1 + r_2$, and set $v_{i+r_2} = v_i$ for $r_1 + r_2 < i \leq d$ to define the vector $\mathbf{v} = (v_i)_{i \leq d}$ and perform ideal reduction in the same manner as for shortest vector ideal reduction except that LLL is used instead of enumeration. In particular, we take $\gamma \in \mathbf{K}$ to be the element corresponding to the first LLL lattice basis vector (with respect to the v-norm) and define $\mathfrak{b}$ so that $\mathfrak{ab} = (\gamma)$. In this case we have

$$N(\mathfrak{b}) \leq 2^{d(d-1)/4} \cdot |\Delta|^{\frac{1}{2}}$$

using the bound on the first vector $\mathbf{b}_1$ of an LLL reduced basis from Theorem 2.5 using $\delta = 3/4$ in c_δ. Note that in practice δ is taken close to 1. This bound is slightly worse than in shortest vector ideal reduction but in practice we expect to generate relations more quickly.

A method of reducing the basis of an ideal was introduced by [Buc90] that provides more randomization at the infinite places than reduction with respect to $||\cdot||$. LLL reduction is done with respect to the v-norm, see equation (2.6).

3.4 Enumeration Relation Generation

The current versions of both `PARI/GP` and `Magma` use approaches based on enumerating elements of $\mathfrak{a}$. In particular, they enumerate the elements corresponding to short vectors in $\sigma(\mathfrak{a})$. This is done for a large number of initial ideals $\mathfrak{a}$.

3.4.1 `PARI/GP`

Instead of computing a single $\mathfrak{b}$ for each initial ideal $\mathfrak{a}$ we can generate many relations for the same $\mathfrak{a}$ using an enumeration algorithm. Using an enumeration algorithm on the lattice $\sigma(\mathfrak{a})$ we find vectors $\mathbf{w}_i$ of small $||\cdot||$ norm. For each $\mathbf{w}_i$ we get a pair $(\mathfrak{b}_i, \gamma_i)$ so that $\mathfrak{ab}_i = (\gamma_i)$. The benefit is that generating initial ideals $\mathfrak{a}$ takes time so using fewer initial ideals can improve the efficiency.

Although this approach does not appear to have been discussed in the literature it has been implemented in `PARI/GP` [All21]. In `PARI/GP` **v2.6.1** which was released in 2013 this enumeration approach replaced the existing LLL ideal reduction. Given an ideal $\mathfrak{a}$ this function implements the Fincke-Pohst algorithm on $\sigma(\mathfrak{a})$ to enumerate small elements.

3.4.2 `Magma`

`Magma`'s class group algorithm originated with the library `KANT` in the 1990s but it has been refined over time. Like `PARI/GP`, its release notes in recent years suggest the implementation is updated every few years.

According to [BF12], in `Magma` versions prior to **v2.18** the relation generation approach was to enumerate a large number of small elements from an ideal $\mathfrak{a}$ that is a power product of prime ideals. The enumeration was done by applying the Fincke-Pohst method to the lattice generated by a $\mathbf{Z}$-basis of $\mathfrak{a}$.

According to the `Magma` release notes version **v2.18** (released Dec 2011) added a sieving approach as an initial step for generating relations. The code was written by one of the authors of [BF12] and so it was most likely similar to the approach described in [BF12]. This sieving ended when the yield of relations dropped below some threshold value and was followed by an approach using a version of Fincke-Pohst enumeration.

According to the `Magma` release notes version **v2.27.6** (released Nov 2022) removed the initial sieving phase. Our assumption is that the relation generation approach is now done entirely by enumeration.

The log generated by running `Magma`'s class group implementation shows it incorporates many optimizations including a large prime variation and Bernstein's batch algorithm for detecting and factoring B-smooths [Ber04]. It also appears to make use of automorphisms of the number field which we speculate is used to improve relation generation. We have done some benchmarking of `Magma`'s algorithm for computing class groups and unit groups against `PARI/GP`'s. These benchmarks show that it is between three and twenty times faster for the number fields discussed in Section 6.1.

`Magma` has different code for quadratic fields. In this context `Magma`'s implementation contains two approaches: enumeration and an SIQS implementation. It switches from the enumeration-based relation generation to the SIQS implementation at a 67-bit (20 digit) discriminant size for quadratic fields [CBFS23b].

3.5 Degree Two Sieving Relation Generation

In line sieving we work with values of a sieving polynomial at points on the "line" $x \in [-E, E]$. Line sieving is the most traditional way to do sieving but sieving can also be done over a two-dimensional region. The univariate sieving polynomial is replaced by its homogenized form in two indeterminates. For example, we can sieve over $x, y \in [-E, E]$. This can be implemented with lattice sieving. The idea is to set up a lattice in this plane and to sieve around the lattice points in order to identify smooths. We refer to [FK05] for further information.

3.5.1 Choosing the Ideal's Norm

From the discussion in Section 2.5 we know that it is important that $N(\mathfrak{b})$ be made small in order to increase the probability that $N(\mathfrak{b})$ is smooth. In the context of sieving, $N(\mathfrak{b})$ is the value of the sieving polynomial ϕ at a point $x \in [-E, E]$, in the case of line sieving, or $x, y \in [-E, E]$ in the case of lattice sieving. We will discuss some work done in the integer factorization context on minimizing the size of the integers being tested for smoothness of Silverman [Sil87] that was adopted by Jacobson in [Jac99b].

Silverman minimizes the size of the integers being tested for smoothness for quadratic sieving in general. This result is relevant to the sieving approach used by [DH84], the MPQS algorithm Silverman gives later in the same paper, and the sieving algorithms for computing class groups that were later given by Jacobson.

In this general sieving setup, relations are found by finding smooth values of a sieving polynomial $Q(x) = A_2x^2 + A_1x + (A_1^2 - n)/(4A_2)$ on the interval $[-E, E]$ where n is the number to be factored and the constant term is chosen so that $\operatorname{disc}(Q) = n$. Silverman then finds the choice of A_2, A_1 that minimize $\sup |Q(x)|$. This is $A_2 = \sqrt{\frac{n}{2E}}$ and $A_1 = 0$ and hence the maximum value of $Q(x)$ over $[-E, E]$ is

$$E\frac{\sqrt{n}}{2\sqrt{2}}. \tag{3.2}$$

We note that although Silverman considers line sieving it is easy to adapt his results to two dimensional sieving.

This result is general in the sense that it allows arbitrary A_2 and A_1 provided that $A_1^2 - n$ is divisible by $4A_2$, Silverman could, alternatively, have derived a bound specifically for his MPQS approach. The improvement in equation (3.2) if this were done would not be large which explains why Silverman chose to give the more general result.

3.5.2 Multiple Polynomials

In the context of the quadratic sieve for integer factorization, an approach using multiple sieving polynomials rather than just one was already given in [DH84]. Silverman [Sil87] improves on the approach in [DH84] in an algorithm called the Multiple Polynomial Quadratic Sieve (MPQS). Like the approach of [DH84] the sieving polynomials are linear transformations of a single polynomial. In MPQS A_2 plays the role of the q in the approach of [DH84] but instead of a prime q Silverman uses a q that is a product of small primes. In the MPQS algorithms given by Silverman we first fix an $A_2 \approx \sqrt{\frac{n}{2E}}$ that is divisible by many small primes which causes there to be many small values of A_1 that satisfy the condition $(A_1^2 - n)/(4A_2)$. Hence there are many sieving polynomials we can use for this choice of A_2.

In the remainder of this section we describe the MPQS algorithm for computing class groups given by Jacobson [Jac99b], which adapts the approach used in [Sil87] to computing class groups of quadratic fields $\mathbf{Q}(\theta) = \mathbf{Q}(\sqrt{D})$. In this approach a subset $Q = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_t\}$ of the factor base is selected. We use an initial ideal $\mathfrak{a}$ of norm A that is a product of elements of Q or their inverses and so that $A = \mathrm{N}(\mathfrak{a}) = \prod_{i=1}^{t} \mathrm{N}(\mathfrak{p}_i)$ is close to the bound (3.2). These ideals are normalized so that they are integral ideals. We set $\mathfrak{a} = (c)S^{\mathbf{v}}$ where $\mathbf{v} \in \{-1, 1\}^t$ and c is the denominator of $S^{\mathbf{v}}$, the smallest integer so that $(c)S^{\mathbf{v}}$ is an integral ideal. Additionally, only one of $\mathfrak{a}$ and $\mathfrak{a}^{-1}$ is used. This gives 2^{t-1} choices for the initial ideal $\mathfrak{a}$.

For any pair of elements ω_0, ω_1 in an ideal $\mathfrak{a}$ the norm form $\phi(X, Y) = \mathrm{N}(X\omega_0 + Y\omega_1)$ gives the values of the norms of $X\omega_0 + Y\omega_1$. Jacobson uses line sieving so Y is set as 1 and the elements are $\omega_0 = A$ and $\omega_1 = (\theta + b)/2$ for $0 \leq b < A$ with b a root of the defining polynomial $X^2 - D \pmod{4A}$. We have that $\mathrm{N}(\omega_1) = (\theta + b)/2 \cdot (-\theta + b)/2 = (-\theta^2 + b^2)/4 = (b^2 - D)/4$ is an integer multiple of A.

Jacobson determines the norm form using the fact that $\mathbf{K}$ is a Galois extension of $\mathbf{Q}$. As a result the norm of an element can be expressed as a product of conjugates. The norm form is

$$\phi(X) = \mathrm{N}(XA + (\theta + b)/2) \quad = (XA + (\theta + b)/2)(XA + (-\theta + b)/2) = A \cdot \left(AX^2 + bX + \frac{b^2 - D}{4A} \right). \tag{3.3}$$

The norm of the initial ideal A is then selected to minimize the largest value of ϕ over the interval $[-E, E]$. He chooses $A = \mathrm{N}(\mathfrak{a}) = \prod_{i=1}^{t} \mathrm{N}(\mathfrak{p}_i)$ so that it is approximately equal to $E^{-1}\sqrt{\frac{|\Delta|}{2}}$. Recall from Section 4.2 that this choice minimizes the size of A. As a result of equation (3.2) the $N(\mathfrak{b})$

values are smaller than

$$E\frac{\sqrt{|\Delta|}}{2\sqrt{2}}$$

in practice.

3.5.3 Self-Initialization

As with MPQS, the self-initialization approach began with work on integer factorization. In 1988 [PST88] introduced self-initialization as an optimization for MPQS. The approach was later named the self-initialization quadratic sieve (SIQS). The authors suggest two approaches to self-initialization. In one of their approaches, they first select a quadratic polynomial f whose discriminant is divisible by n the number to be factored. This condition is related to their goal of factoring integers. Next they choose K as a product of l primes $\prod p_i$ that split in $\mathbf{Q}[X]/(f)$. They consider the solutions u of $f \equiv 0 \pmod{K}$ and for each solution u they define the polynomial $g_u(X) = \frac{1}{K}f(u + XK)$ which can be line sieved over $[-E, E]$. This g_u is the sieving polynomial of the pair $[K, \theta - u]$ as $g_u = \frac{1}{K}\mathrm{N}(XK + \theta - u)$ by equation 2.2. There are 2^l solutions u and so this gives 2^l polynomials. They indicate that a self-initialization approach can be used in order to obtain the sieve initialization data for these polynomials. A fast implementation of self-initialization was later described [Con97] who found it to be twice as fast as MPQS.

Jacobson's approach is not the only approach to self-initialization in quadratic number fields, another approach has been given by [Kle16]. The PhD thesis [Luo24] adapts the approach of [Kle16] to higher degree number fields. In the sequel we focus on an adapting the ideas of Jacobson's approach to work in higher degrees so we will limit our discussion of self-initialization to Jacobson's approach. In the rest of this section we describe the SIQS algorithm of Jacobson [Jac99b].

Recall that this approach is defined in the field $\mathbf{Q}(\theta) = \mathbf{Q}[X]/(f)$ where $f = X^2 - D$. The element $(\theta + b)/2$, the norm form ϕ and the roots of ϕ mod the rational primes p_i are the data we need in order to be able to perform sieving. We want to choose a sequence of ideals so that we can quickly iterate over this sieving data. The Chinese remainder theorem is used to iterate quickly through 2^{t-1} roots b_i of the defining polynomial modulo $4A$. In other words these b_i are solutions of $b_i^2 \equiv D \pmod{4A}$.

In the case $|\Delta| \equiv 0 \pmod{4}$ the Chinese remainder theorem can be applied directly to obtain the value of the b_i modulo the p_j. We have that there exist integers B_j for $1 \leq j \leq t$ such that $B_j \equiv 0 \pmod{p_k}$ if $k \neq j$ and $B_j^2 \equiv D \pmod{p_j}$ otherwise. Any combination of the form $B = \pm B_1 \pm B_2 \pm \cdots B_{t-1} + B_t$ is a root of $X^2 - D \pmod{p_j}$. The sign of B_t is fixed so that only one of $B \pmod A$ and $-B \pmod A$ is constructed. Then $B_j = (A/p_j)((A/p_j)^{-1}t_j \pmod{p_j})$ where t_j is a root of the defining polynomial $\pmod{p_j}$. If $|\Delta| \equiv 1 \pmod 4$ then B_j can be determined with a variation of this idea.

We denote the roots of f modulo A by $P_1, \ldots, P_{2^{t-1}}$. We use a Gray code ordering on these roots. By the definition of a Gray code P_{i+1} and P_i differ at exactly one prime p_k in $\{p_j\}_{j=1}^t$. In other words $P_{i+1} - P_i$ is a multiple of B_k. We take $P_1 = B_1 + \ldots + B_t$. Each P_i is associated to a vector $\mathbf{u} \in \{-1, 1\}^t$. The ordering on the $\mathbf{u}$ is defined $\mathbf{u}_{i+1} = \mathbf{u}_i + (0, \ldots, 0, 2(-1)^{\lceil i/2^{\nu_2(2i)} \rceil}, 0, \ldots, 0)$ where the index of the nonzero entry is $\nu_2(2i)$. Then $P_i = \mathbf{u}_i \cdot (B_j)_{j=1}^t$ and so the ordering on the P_i is defined $P_{i+1} = P_i + 2(-1)^{\lceil i/2^{\nu_2(2i)} \rceil} B_{\nu_2(2i)}$.

Using P_i it is easy to compute ϕ_i using equation (3.3). If $p_j \nmid A$ then the roots of ϕ_i modulo p_j are related by

$$r_{i+1,j} \equiv r_{i,j} - 2Q^{-1}(-1)^{\lceil i/2^{\nu_2(2i)} \rceil} \pmod{p_j}.$$

As there are not too many primes that can divide A it is not expensive to compute the roots modulo such primes separately.

3.6 Higher Degree Sieving Relation Generation

After the development of the Number Field Sieve (NFS) for integer factorization, it was observed that for defining polynomials f like those used in NFS, polynomials of small height, a very similar sieving approach to NFS would work for computing class groups of $\mathbf{Q}[X]/(f)$. This was first suggested in [BD91]. To illustrate the main point if we take $\omega_0 = 1$ and $\omega_1 = -\theta$ (the choice in the textbook Number Field Sieve) the sieving polynomial of the pair $[\omega_0, \omega_1]$ is $f(X)$. We can sieve this polynomial in the same manner as in the quadratic case. Hence sieving with these elements amounts to sieving the defining polynomial. If the defining polynomial has small height then this approach has the potential to be effective.

A more general idea described in [Nei02] is to take sieving pairs (ω_0, ω_1) for $\omega_0, \omega_1 \in \mathfrak{a}$ so that the heights of the minimal polynomials of ω_0 and ω_1 is small. The polynomial $\phi = \mathrm{N}(\omega_0 X + \omega_1 Y)/\mathrm{N}(\mathfrak{a})$ in $\mathbf{Z}[X, Y]$ satisfies $\mathrm{N}(\mathfrak{b}) = |\phi(x, y)|$ for all $x, y \in \mathbf{Z}$ and $\omega_0, \omega_1 \in \mathfrak{a}$ where $\mathfrak{b}$ is the ideal satisfying $\mathfrak{ab} = (\omega_0 x + \omega_1 y)$. We call this approach the **sieving with elements of small height**.

In order to compute ϕ an interpolation method is used. Values of $\mathrm{N}(\omega_0 x + \omega_1)/\mathrm{N}(\mathfrak{a})$ are computed for $d+1$ values of x. The norm form is then computed using Lagrange interpolation. A downside of the interpolation approach is that it must be reused for each pair of sieving elements and so becomes expensive as the number of sieving polynomials grows.

The author of [Nei02] gives timings of his sieving algorithm. In Table 5.2 he computes the class group of cubic fields with a defining polynomial that has large height. These heights are approximately $|\Delta|^{2/3}$. In Anhang A and Anhang B he gives timings for number fields that have defining polynomials of small height. These heights are approximately $|\Delta|^{1/(2(d-1))}$ for $d = 3, 4, 5, 6$. The performance of the algorithm is far superior for the number fields with defining polynomials of small height.

In [BF12] the approach of [Nei02] is modified and other new techniques are introduced for computing class groups. The authors of [BF12, Section 4.1] generate α, β using a multi-step process. They first define the embedding $\psi_{\vec{a}} : \mathfrak{a} \to \mathbf{R}^d$ with $\alpha \to (a_1 \log|\alpha|_1, \ldots, a_d \log|\alpha|_d)$ where $\mathbf{a} = (a_1, \ldots, a_d)$ is a vector of real numbers. They do reduction with a norm associated to this embedding. The main steps are as follows:

- Fix an integral ideal $\mathfrak{a}$.
- Define a vector of random coefficients $\mathbf{a}$.
- Define the lattice $\Delta_{\mathbf{a}}$ generated by $\alpha \in \mathfrak{a}$ under an embedding with respect to a certain norm that is defined using $\mathbf{a}$.
- Choose α, β so that α and β are the first two elements of the LLL-reduced basis for $\Delta_{\mathbf{a}}$.
- Return the norm form $P_{\alpha,\beta}(X, Y) = N(x\alpha + y\beta)$.

Importantly, this process should be used with $\mathfrak{a} = (1)$ to generate most of the relations. If this is not sufficient then other $\mathfrak{a}$ can be used [Bia23].

The approach of defining a lattice and doing reduction with respect to it resembles the LLL ideal reduction approach. A key difference is that the Biasse-Fieker approach requires far fewer LLL reductions. Another difference between these approaches is the lattice and norm that are used for the reduction. Buchmann's approach introduced the idea of using a random vector to define the v-norm (see equation (2.6)). This norm is important for giving randomization at the Archmidean components (see [CDO97, Algorithm 6.5.9 Remark 2]). The lattice and norm used by [BF12] are intended for the same purpose but differ in that a logarthmic lattice is used.

There are two main aspects to the performance of a relation generation algorithm. First, the size of the norms $\mathrm{N}(\mathfrak{b})$. Second, the time needed to generate the relations which in this case is simply the time required to test $\mathrm{N}(\mathfrak{b})$ for B-smoothness. Like [Nei02], the authors focus mainly on the speed of relation generation rather than on determining how large $\mathrm{N}(\mathfrak{b})$ is for their approach. The authors include the requirement that "our choices of α and β yield polynomials with small coefficients." In other words, the resulting sieving polynomials must have small height. What the height should be, or why the steps described above should produce polynomials with small height, is not clear.

In addition, they suggest applying the special-q sieving used in implementations of the Number Field Sieve to the problem of computing class groups. Specifically, they suggest using a special-q approach for prime q with $q \leq B$ so that f has a root modulo q. For these primes, the elements q and $\theta - s$ where s is a root of f modulo q are elements of the degree one prime ideal $(q, \theta - s)$. The vectors $(q, 0), (-s, 1)$ of their coordinates generate a lattice of elements with norm divisible by q. A Gaussian reduction is then used to obtain a reduced basis ω_0, ω_1 of this lattice. The norm form $\mathrm{N}(X\omega_0 + Y\omega_1)/q$ is straightforward to compute and to sieve. We refer to [BF12] for details.

Once Λ' has full rank they suggest using methods they call saturation on the class group and saturation on the unit group to expand the lattice of relations. In this method the elements γ_i in the relations $(\mathbf{v}_i, \gamma_i)$ are viewed as S-units with $S = \mathrm{FB}_{\mathbf{K}}$. The set of all the γ_i generates a subgroup $\mathcal{U}$ of the S-unit group. Fixing a prime $p \mid [\mathcal{U}_S : \mathcal{U}]$ the goal is to find some $\gamma \in \mathcal{U}_S \backslash \mathcal{U}$ with γ represented as a power-product such that γ has a pth root α in $\mathcal{U}_S$ but not in $\mathcal{U}$. If α can be determined as a power-product of the γ_i then $(\mathbf{v}, \alpha)$ where $\mathbf{v}$ is the 0 vector is a relation which expands the lattice.

In [BF12] the authors implemented their ideas and give some data supporting the use of sieving. They write "For each size d, we drew at random 10 number fields with discriminant satisfying $\log_2(|\Delta|) = d$." Presumably, these polynomials were constructed by choosing coefficients from a bounded interval. As we discuss in Section 5.3 this leads to polynomials of height $O(|\Delta|^{1/(2(d-1))})$. This is a very small height. They optimize for speed by combining the sieving with elements of small height with the enumeration approach used in `PARI/GP`. Specifically, they try to produce enough relations with their sieving approach so that the rank of M is 97% of $|\mathrm{FB}_{\mathbf{K}}|$. All further relations are found with the enumeration approach in `PARI/GP`. In Table 1 they give data for the performance of their algorithm. In number fields of degree up to 6 they consistently find it is faster than `PARI/GP`. The speedup of their algorithm over `PARI/GP`'s is largest in degree three and drops for each successive degree. In other words, the effectiveness of sieving decreases as the number field's degree grows larger. For example, for the quartic fields of ~180 bits they tested, their algorithm was ~26 times faster than `PARI/GP`. Their results show that, for the class of defining polynomials they tested, their hybrid sieving approach performs an order of magnitude better than the ideal reduction approach in `PARI/GP`.

Table 2 is somewhat different. It shows the timings for quadratic polynomials of the form $f = X^2 + 4 \cdot (10^n + 1)$. These polynomials have height $O(|\Delta|)$. This is very large compared with the presumed $O(|\Delta|^{1/d})$ height in Table 1. In this case their timings were worse than `PARI/GP`. For example, for the ~54 digit (~180 bits) quadratic field, their algorithm was ~9 times slower than `PARI/GP`. Differences in the height of the defining polynomials in the two tables may explain why Biasse-Fieker's results are much better for the timings in Table 1 than for the timings in Table 2.

An analysis of this algorithm is beyond the scope of this thesis but we will briefly comment on the special case we gave at the start of this section. This saves us from dealing with the complexity of the LLL reduction. We take $\mathfrak{a} = (1)$ We set $\alpha = 1$ and $\beta = \theta$ then by equation (2.2) we have

$$P_{\alpha,\beta}(X, Y) = Y^d f(XY^{-1})$$

so $H(P_{\alpha,\beta}) = H(f)$ and the goal is to find relations by sieving with elements of small height. In order to minimize the height it would be beneficial to pass our initial defining polynomial to an algorithm that tries to output a defining polynomial of small height before sieving this polynomial. Unfortunately, we do not know of any algorithm that can take an arbitrary defining polynomial and output a defining polynomial of height $H(f) = O(|\Delta|^{\frac{1}{2}})$ even in exponential time.

Example 3.1. *Let us consider an example of the size of norm that might be obtained if we happen to be given a defining polynomial of height approximately $|\Delta|^{\frac{1}{2}}$. Sieving the intervals $X \in [-E, E]$ and $Y \in [-E, E]$ the size of the norms being tested for smoothness are*

$$(d + 1)|\Delta|^{\frac{1}{2}} E^d.$$

For small d this is comparable with the norms $N(\mathfrak{b})$ produced by Buchmann's approach which are upper bounded by $2^{d(d-1)/4} \cdot |\Delta|^{\frac{1}{2}}$ (see Section 3.3).

Regarding implementations of sieving-based approaches, we do not know of any publicly available implementations. Prior versions of `Magma` used a sieving approach for some cases but this is no longer the case.

Related Work

Other approaches for sieving relation generation have been suggested for $d \geq 2$ that focus primarily on time complexity results rather than on practical performance. [Bia14a] gives an algorithm of heuristic subexponential time complexity bounded by $L_{|\Delta|}[1/3]$ for families of number fields with defining polynomials of small height. [Gél18b] gives a variation of the sieving approach of [Bia14a], broadens somewhat the families of number fields the approach applies to, and gives a more detailed analysis of the time complexity, including both the first and second term in the L-notation,

Chapter 4

Improvements to Sieve-based Relation Generation Applied to Quartic Fields

Contents

This chapter presents the new results of this thesis. First, in Section 4.1 we consider the size $||f||$ of defining polynomials f. We give a definition of a **reduced** defining polynomial and show some families of these polynomials with different values of λ.

In Section 4.2 we consider ideals $\mathfrak{a}$ that are products of degree one prime ideals that do not divide $[\mathcal{O}_{\mathbf{K}} : \mathbf{Z}[\theta]]$. For $\mathfrak{a}$ of this form the ideals we test for smoothness have the form $(xA + y\omega_1)$ for integers x and y in the interval $[-E, E]$ where A, ω_1 are in $\mathfrak{a}$. In order to make the numbers $g(x, y) = \mathrm{N}(xA + y\omega_1)/A$ being tested for smoothness have a high probability of being smooth we try to keep $g(x, y)$ as small as possible for a given E. This is done by choosing A so that $\mathrm{N}(xA + y\omega_1)/A$ is small. This choice is important because the size of the norms of elements found by sieving is determined by this choice and is a primary driver in the performance of the overall algorithm. The size of the norm is a crucial factor in the speed of relation generation of any index-calculus algorithm. The importance of the size of the integers being tested for smoothness can be seen, for example, in factoring a number n. The performance gap between the quadratic sieve and the general number field sieve comes from the fact that in the former the integers that are tested for smoothness have roughly $n^{\frac{1}{2}+\epsilon}$ versus $\exp(c \log(n)^{2/3} \log(\log(n))^{1/3})$ for the latter, for small constants ϵ and c [Len17].

In Chapter 3 we discussed the sieving approaches of MPQS and sieving with elements of small height. In the MPQS approach we select a pair of elements A, ω_1 in an ideal $\mathfrak{a}$, where $A \in \mathbf{Z}$ and $\omega_1 \in \mathcal{O}_{\mathbf{K}}$, and test elements in the set $T = \{xA + y\omega_1\}_{(x,y)\in S}$ for B-smoothness using their sieving polynomial. In this chapter we describe an analogue of this approach and then we discuss how the approach can be made more efficient using an analogue of the self-initialization approach in SIQS.

In Sections 4.3 and 4.4 we give a description of self-initialization that is applicable to quartic fields. The description of this approach is similar to the one in Sections 3.5.2, and Sections 3.5.3 respectively. In Section 4.3 we give a sieving approach utilizing multiple sieving polynomials which permits self-initialization to be added on as an optimization. In Section 4.4 we present our approach self-initialization which generalizes the approach used in the quadratic case to arbitrary degree number fields.

To make these results applicable beyond the context of quartic fields, they are presented for arbitrary field degrees d. We also interpret the results in the context of a quartic fields as our implementation and testing are focused on this case.

4.1 Reduced Defining Polynomials

In the quadratic case, the MPQS algorithm depends on a certain choice of defining polynomial. In this case Jacobson [Jac99b] uses the defining polynomial

$$f = X^2 - t.$$

One notable characteristic of this defining polynomial is that for each root r we have $|r| = t^{\frac{1}{2}}$. Recall the definition of $||f||$ from Section 2.7, which quantifies the magnitude of the polynomial's roots. As both roots have the same size, $||f||$ is very small amongst all possible defining polynomials for $\mathbf{Q}(\theta)$.

In number fields of higher degree we choose a defining polynomial so that $||f||$ is small. The choice of defining polynomial is important because the optimal sieving norm A is a function of $||f||$. We derive this norm in Section 4.2.

Definition 4.1. Let f be an irreducible degree d monic polynomial that defines a number field with discriminant Δ. We define λ as the real number so that

$$||f|| = 2^d|\Delta|^\lambda.$$

We say f is **λ-reduced** if $\lambda \leq \frac{1}{2(d-1)}$.

A polynomial f that meets the criteria for being λ-reduced, where the precise value of λ is not important, can simply be referred to as **reduced**.

Our requirement is that in practice `polredabs` polynomials be λ-reduced. The main idea of `polredabs` is to apply LLL reduction to an integral basis of the maximal order of the number field. Regarding the choice of the constant 2^d, our choice is inspired by the constant c_δ that appears in bounds on the basis elements that are produced by LLL. We have not tied Definition 4.1 to the `polredabs` algorithm because the approach we describe in the sequel does not require the defining polynomial of the number field to have been produced with `polredabs`. After giving some examples of defining polynomials with different λ values, we provide some computational evidence to justify that `polredabs` polynomials are reduced.

Example: Simplest Quartic Fields

The simplest fields introduced by Daniel Shanks [Sha74] are known for their elegant properties. In degree four the simplest quartic fields have similar properties. Such a field has is defined by

$$f = X^4 - tX^3 - 6X^2 + tX + 1.$$

for an integer t that we can specify to be nonnegative because of the symmetry of the coefficients. This polynomial is reducible exactly when $t^2 + 16$ is a square which only happens at $t = 0$ and $t = 3$. Where f defines a totally real cylic field. Results on the simplest quartic fields are given in the PhD thesis [Laz89].

We will show the defining polynomials of the simplest quartics fields satisfy the size property with $\lambda = \frac{1}{6}$. There are four possibilities for the field's discriminant:

- $(t^2 + 16)^3$,
- $(t^2 + 16)^3/4$,
- $(t^2 + 16)^3/16$,
- $(t^2 + 16)^3/64$.

see [Laz89, Section 2.3]. This means

$$(t/2)^6 \leq ((t/2)^2 + 4)^3 \leq |\Delta| \text{ so } t \leq 2|\Delta|^{1/6}. \tag{4.1}$$

The roots are

$$\epsilon_{1,2,3,4} = \pm\frac{\sqrt[4]{t^2+16}\sqrt{\sqrt{t^2+16} \pm t}}{2\sqrt{2}} \pm \frac{\sqrt{t^2+16}}{4} + \frac{t}{4}$$

for t not 0 or 3 where the second and third signs agree [Laz89]. We can order these roots by the size of $|\epsilon_i|$ with $|\epsilon_1|$ being the largest. We find ϵ_1 by taking all the signs positive yielding $\epsilon_1 = \frac{\sqrt[4]{t^2+16}\sqrt{\sqrt{t^2+16}+t}}{2\sqrt{2}} + \frac{\sqrt{t^2+16}}{4} + \frac{t}{4}$. An upper bound on ϵ_1 can be found using the bound on the largest root from equation (2.1) which gives $\epsilon_1 \leq 1 + \max\{|-t|, |6|, |t|, 1\}$. So we have $\epsilon_1 \leq 1 + t$ for $t \geq 6$.

The other three roots all have very small absolute value. One way to see this is to lower bound ϵ_1 and then use the relationship between the polynomial's height and its Mahler measure $M(f)$. We can lower bound ϵ_1 by dropping all +16 terms and this shows $t \leq \epsilon_1$ for all t. We have $M(f) \leq \sqrt{d+1}H(f)$ for any polynomial f (see [GJ16, Section 2.2]) so in our case

$$t \cdot \prod_{i=2}^{4} |\epsilon_i| \leq \prod_{i=1}^{4} \max(1, |\epsilon_i|) = M(f) \leq \sqrt{d+1}H(f) = \sqrt{5} \cdot t \text{ for } t \geq 6. \tag{4.2}$$

This gives $t \cdot |\epsilon_2|^3 \leq \sqrt{5} \cdot t$ so $|\epsilon_i| \leq \sqrt[6]{5}$ for $i \in \{2, 3, 4\}$.

We have $||f|| \leq (\epsilon_1^2 + 3\epsilon_2^2)^{\frac{1}{2}}$ Combining these upper bounds for $t \geq 6$ we have

$$\begin{aligned} ||f|| &\leq ((1+t)^2 + 3\sqrt[3]{5})^{1/2} \\ &\leq ((t+1)^2 + t)^{1/2} \text{ as } 3\sqrt[3]{5} < 6 \leq t \\ &\leq (t^2 + 3t + 1)^{1/2} < (16t^2)^{1/2} = 8t. \end{aligned}$$

We also have $t \leq 2|\Delta|^{1/6}$ for $t \geq 6$ from equation (4.1). So the defining polynomials for simplest quartic fields are reduced for $t \geq 6$.

Polynomials With Bounded Coefficients

The defining polynomials for simplest quartic fields are contained in a larger class of defining polynomials with **bounded coefficients**. The polynomials tend to be λ-reduced with $\lambda = \frac{1}{6}$ in practice. In particular, consider polynomials whose coefficients have absolute value bounded by $2^{k/(2(d-1))}$ and so that the absolute value of the largest coefficient is close to $2^{k/(2(d-1))}$ where k is some integer. For instance, the defining polynomials of the simplest quartic fields are an example of bounded coefficient polynomials with k defined by $t = 2^{k/6}$.

In practice the number fields defined by such polynomials have discriminant approximately 2^k. We discuss these polynomials further in Section 5.2.

Example: Pure Fields

A **pure field** is a number field defined by

$$f = X^d - t$$

where t is not divisible by a dth power. This defines the number field $\mathbf{Q}(\sqrt[d]{t})$.

The discriminant of a pure quartic is computed in [Fun84]. Let $t = ab^2c^3$ where

- $a \neq 1$,
- b and c are squarefree and coprime and $b > 0, c > 0$,
- $|a| \leq c$ if a is odd,
- c is odd, and
- $t \neq 4$.

In [Fun84, Corollary 1] the field's discriminant is computed by determining an integral basis. Its value depends on the residue of t modulo 32.

$$\Delta = \begin{cases} -2^2a^3b^2c^3 & \text{if } t \equiv 1(\text{mod} 8), \quad 28(\text{mod} 32), \\ -2^4a^3b^2c^3 & \text{if } t \equiv 4(\text{mod} 16), \quad 5(\text{mod} 8), \quad 12(\text{mod} 32), \\ -2^8a^3b^2c^3 & \text{if } t \equiv 2(\text{mod} 4), \quad 3(\text{mod} 4). \end{cases} \tag{4.3}$$

To construct some examples of λ-reduced defining polynomials we work with $t = ab^2$ for integers a, b where $a^\ell \leq b \leq 2a^\ell$ for $0 \leq \ell$ and the conditions of [Fun84] are met: $a \neq 1, b > 0, ab^2 \neq 4$. To be as concrete as possible, we also require that $ab^2 \equiv 1 \pmod 8$. Then

$$2^2a^{3+2\ell} \leq |\Delta| \tag{4.4}$$

$$a \leq 2^{-2/(3+2\ell)}|\Delta|^{\frac{1}{3+2\ell}} \tag{4.5}$$

so $a \leq 2^{2/(3+2\ell)}|\Delta|^{1/(3+2\ell)}$. One of the roots is $t^{1/4}$ and the others are equal to it up to a root of unity. Thus for every root r we have $|r| = t^{1/4}$. Hence

$$|r| = t^{1/4} = (ab^2)^{1/4} \leq (2a^{1+2\ell})^{1/4} \leq w|\Delta|^{T(\ell)}. \tag{4.6}$$

where $T(\ell) = \frac{1+2\ell}{4(3+2\ell)}$ and

$$w = (2 \cdot 2^{2(1+2\ell)/(3+2\ell)})^{\frac{1}{4}}$$

for every root r. Determining $||f||$ is then done by substituting this bound in the definition $||f|| \leq (4w^2 \cdot |\Delta|^{\frac{1+2\ell}{2(3+2\ell)}})^{1/2} = 2w|\Delta|^{\frac{1+2\ell}{4(3+2\ell)}}$.

On $\ell \in [0, \infty)$ the function $T(\ell)$ has a minimum of $\frac{1}{12}$ and is a strictly increasing function that approaches a horizontal asymptote of $\frac{1}{4}$. On this interval we can compute the largest value of ℓ for which $X^4 - ab^2$ is reduced by solving $\frac{1+2\ell}{4(3+2\ell)} = \frac{1}{6}$. The solution is $\ell = \frac{3}{2}$. We see that $w = (2 \cdot 2^{2(1+2\ell)/(3+2\ell)})^{\frac{1}{4}} \leq 2^3$ on this interval so $X^4 - ab^2$ is a reduced defining polynomial for $\ell \leq \frac{3}{2}$. For $\ell > \frac{3}{2}$ we could simply apply the `polredabs` algorithm to get a $\frac{1}{6}$-reduced defining polynomial.

In this interval $\ell = 0$ gives the best possible $||f||$ and $\ell = 3/2$ gives the worst possible and the exponents of $|\Delta|$ are respectively $\frac{1}{12}$ and $\frac{1}{6}$. In Chapter 6 we give some computations for a family of defining polynomials with $\ell = 0$ and also for a family where ℓ is chosen to make the exponent on $|\Delta|$ the mean of the exponents for the best and worst case: $\frac{1}{2}(\frac{1}{6} + \frac{1}{12}) = \frac{1}{8}$ this happens when $\ell = \frac{1}{2}$.

In summary for

$$\ell = 0 \text{ we have } ||f|| \leq 2w|\Delta|^{\frac{1}{12}} \text{ and for}$$
$$\ell = \frac{1}{2} \text{ we have } ||f|| \leq 2w|\Delta|^{\frac{1}{8}} \text{ and for}$$
$$\ell = \frac{3}{2} \text{ we have } ||f|| \leq 2w|\Delta|^{\frac{1}{6}}.$$

`polredabs` Polynomials

For degree four polynomials λ is defined by $||f|| = 2^4|\Delta|^{\lambda}$. We can approximate the average λ for degree four `polredabs` polynomials that define number fields of discriminant up to some bound. The `LMFDB` has a complete database of `polredabs` defining polynomials for quartic fields of discriminant up to 4 million. We tested $n = 1024$ defining polynomials of quartic fields selected uniformly at random from this database and found all of them to be reduced. In other words $\lambda \leq \frac{1}{6}$ for all these polynomials. This gives us some assurance that `polredabs` polynomials are usually reduced in practice for $d = 4$.

We also approximate the average λ over these polynomials. To do this we compute

$$D^* = \frac{1}{m} \sum_{i=0}^{m-1} \log |\Delta| \text{ and} \tag{4.7}$$

$$S^* = \frac{1}{m} \sum_{i=0}^{m-1} (\log ||f_i|| - 4) \tag{4.8}$$

for the same $m = 1024$ defining polynomials f_i as above. The average λ is equal to S^*/D^* and for our polynomials we found $S^*/D^* \approx 0.1477$. This is quite close to $1/6$

$$1/6 - \lambda^* \approx 1/6 - 0.1477 \approx 0.0189.$$

This gives some evidence that λ is close to $1/6$ for quartic `polredabs` defining polynomials. We have seen certain families of defining polynomials with $||f||$ significantly smaller than $|\Delta|^{\frac{1}{2(d-1)}}$. In the sequel we will see that the sieving approach we will give is much more efficient for these polynomials. Unfortunately, this data suggests that such defining polynomials are rare.

4.2 Choosing the Ideal's Norm

We consider how to choose A so that the ideals $\mathfrak{b} = (xA + y\theta - yP)$ have small norm. Here P is an arbitrary integer but later in this chapter we will specialize to an integer P that is analogous to the P we used in Section 3.5. We want to choose A so that

$$\sup\{\mathrm{N}(\mathfrak{b})\} = \sup\left\{\mathrm{N}(xA + y\theta - yP)/A\right\}_{x,y=-E}^{E} \tag{4.9}$$

is minimized as a function of $|\Delta|$. In practice, the maximum value is achieved when x and y are near the endpoints of the interval $[-E, E]$ so we can find an approximate solution by minimizing $\sup\{\mathrm{N}(EA + E\theta - EP)/A\}$. As E is constant $\mathrm{N}(EA + E\theta - EP) = E^d\mathrm{N}(A + \theta - P)$. Note that in practice E is small compared with $|\Delta|$.

The following result gives a choice of A that makes $\mathrm{N}(EA + E\theta - EP)$ and hence $\mathrm{N}(\mathfrak{b})$ small.

Theorem 4.2. *Let d be fixed, and let A be the closest integer to $d^{\frac{-1}{2}}(d-1)^{-1}||f||$. As $|\Delta|$ tends to infinity, the following asymptotic relation holds: $N(EA + E\theta - EP)/A = O(E^d||f||^{d-1})$.*

Proof.

$$\begin{aligned}
|\mathrm{N}(A + \theta - P)|/A &\leq d^{-\frac{d}{2}}(||A + \theta - P||)^d \cdot A^{-1} \text{ by the AM-GM inequality} \\
&\leq d^{-\frac{d}{2}}(||(A - P) + \theta||)^d \cdot A^{-1} \\
&\leq d^{-\frac{d}{2}}\left(d^{\frac{1}{2}}(A - P) + ||f||\right)^d A^{-1} \text{ by the triangle inequality} \\
&\leq d^{-\frac{d}{2}}\left(d^{\frac{1}{2}}A^{\frac{d-1}{d}} + ||f||A^{\frac{-1}{d}}\right)^d \text{ as } 0 \leq P < A
\end{aligned}$$

We minimize $h = d^{\frac{1}{2}} A^{\frac{d-1}{d}} + ||f|| A^{\frac{-1}{d}}$. Rearranging $\frac{\partial h}{\partial A} = 0$ we get

$$\begin{aligned} \frac{d^{\frac{1}{2}}(d-1)}{d} A &= \frac{1}{d}||f|| \\ A &= d^{\frac{-1}{2}}(d-1)^{-1}||f|| \end{aligned}$$

We can evaluate

$$d^{-\frac{d}{2}} \left(d^{\frac{1}{2}} A^{\frac{d-1}{d}} + ||f|| A^{\frac{-1}{d}} \right)^d$$

with $A = d^{\frac{-1}{2}}(d-1)^{-1}||f||$ and we find $|\mathrm{N}(A+\theta-P)| = O(||f||^{d-1})$ from which the result follows. $\square$

Theorem 4.2 does not depend on the number of sieving polynomials that we use and so a single polynomial could be used. It could be used when sieving any ideal that is a product of degree one prime ideals and whose norm is coprime to the index. For this reason it is a separate contribution from the self-initialization relation generation approach we give in Section 4.3-4.4.

Reduced Defining Polynomials Cases: Special and General

In Section 2.7 we gave computational evidence that degree four polynomials that are the output of `polredabs` are reduced with $\lambda = \frac{1}{6}$. Theorem 4.2 shows that

$$\mathrm{N}(EA + E\theta - EP)/A = O(E^d |\Delta|^{\frac{1}{2}})$$

under this heuristic.

The polynomials $f = X^d - t$ are typically left unchanged by `polredabs` because all the roots are the same size. As shown in Section 4.1 these polynomials are reduced with $\lambda = \frac{1}{12}$ when t is squarefree. Thus, there are `polredabs` polynomials where λ is at least as small as $\lambda = \frac{1}{12}$. When we have a defining polynomial with λ a lot smaller than $\frac{1}{6}$ it is much faster to generate enough relations. For $\lambda = 1/12$ the algorithm gives norms that are $O(E^d|\Delta|^{\frac{1}{4}})$ rather than $O(E^d|\Delta|^{\frac{1}{2}})$ when $\lambda = 1/6$.

Thus reduced defining polynomials have an associated λ that can vary between $\frac{1}{12} = \frac{1}{4(4-1)} \leq \lambda \leq \frac{1}{2(4-1)} = \frac{1}{6}$ and we conjecture that λ is close to $\frac{1}{6}$ for most such polynomials. For the purpose of discussing the performance of our algorithm it is useful to put defining polynomials into two categories according to the size of λ. We call the situation when λ is much smaller than $\frac{1}{6}$ the **special** case and otherwise we say we are in the **general** case.

Choice of Sieving Approach

Theorem 4.2 assumes that 2-dimensional sieving is being used for relation generation. The same method can be used to give a bound if line sieving is used instead. Let A_{2d} be the value of A in Theorem 4.2 and A_{1d} be the corresponding result for line sieving. Then the values of A_{1d}, A_{2d} are related by $A_{\mathrm{1d}} = E^{-1}A_{\mathrm{2d}}$. When the discriminant is small, it is clear that the target norm of $d^{\frac{-1}{2}}(d-1)^{-1}||f||$ for 2-dimensional sieving will be small. The target norm for line sieving $d^{\frac{-1}{2}}(d-1)^{-1}||f||/E$ is smaller still. As we will discuss in the next section, we want $d^{\frac{-1}{2}}(d-1)^{-1}||f||$ to be large so that we can use the largest number of sieving polynomials possible. This makes 2-dimensional sieving the more practical choice.

This approach serves as an analogue to the MPQS method of Jacobson, and is distinct from existing sieving approaches in the literature in several key ways: First, our approach uses a number of different initial ideals $\mathfrak{a}$ rather than sieving primarily in the ideal (1). Second, our approach does not require using the LLL algorithm to try to find sieving elements of small height. Third, sieving elements are constructed with the Chinese remainder theorem, which is more efficient than using LLL. Fourth, due to the use of Theorem 4.2 we can accurately estimate the size of the norms this method produces. We will discuss the differences between our approach and existing approaches in more detail at the end of this chapter.

4.3 Sieving With Multiple Polynomials

To sieve with multiple polynomials we choose a set of t primes from $\mathrm{FB}_{\mathbf{Z}}$ with properties analogous to the quadratic case. We work with a subset of t primes chosen from $\mathrm{FB}_{\mathbf{Z}}$ so that there is a pair of distinct degree one prime ideals $\mathfrak{p}_j^{(0)}$ and $\mathfrak{p}_j^{(1)}$ above p and so that $p \nmid [\mathcal{O}_{\mathbf{K}} : \mathbf{Z}[\theta]]$. By Theorem 2.6 we know that the two degree one prime ideals can be written

$$\mathfrak{p}_j^{(0)} = (p, \theta - r_j^{(0)}) \text{ and } \mathfrak{p}_j^{(1)} = (p, \theta - r_j^{(1)})$$

for some integers $r_{j,0}$ and $r_{j,1}$ that are roots of f modulo p. Setting

$$S_{\mathbf{K}} = \{\mathfrak{p}_0^{(0)}, \mathfrak{p}_0^{(1)}, \mathfrak{p}_1^{(0)}, \mathfrak{p}_1^{(1)}, \ldots, \mathfrak{p}_{t-1}^{(0)}, \mathfrak{p}_{t-1}^{(1)}\}$$

when we choose the t primes from $\mathrm{FB}_{\mathbf{Z}}$

$$A = \mathrm{N}(\prod_{i=0}^{t-1} \mathfrak{p}_i^{(0)}) = \prod_{i=0}^{t-1} p_i$$

is close to the bound given in Theorem 4.2. The particular choice of this subset is a detail of our implementation, and so we discuss it in Section 4.5.

As in the quadratic case, we want to generate a set of relations so that their ideals are divisible by an ideal of the form $\mathfrak{a} = \prod_{i=0}^{t-1} \mathfrak{p}_j^{(e_j)}$ where e_j is either 0 or 1 without computing $\mathfrak{a}$ by the time-consuming approach of repeated ideal multiplication. Instead, we do this by finding a pair of elements contained in $\mathfrak{a}$. Then any linear combination of these elements generates a principal ideal that is divisible by $\mathfrak{a}$. For j from 0 to $t-1$ we let P be the unique integer so that

$$P \equiv r_j^{(e_j)} \pmod{p_j} \tag{4.10}$$

and $0 \le P < A$. The Chinese remainder theorem allows us to compute P efficiently. Then $\theta - P \equiv 0 \pmod{\mathfrak{p}_j^{(e_j)}}$ so $\theta - P$ is an element of $\mathfrak{a}$. Thus for any integers x, y the element $xA + y\theta - yP$ is also in $\mathfrak{a}$. In other words, $(xA + y\theta - yP)$ is divisible by $\mathfrak{a}$.

We recall the definition of the sieving polynomial (Definition 2.10). Our sieving polynomial is $A^{-1}\phi(X,Y) = A^{-1}\mathrm{N}(XA + Y\theta - YP) = \mathrm{Res}_\theta(XA + Y\theta - YP), f(\theta))$. These polynomials over all values of P and the roots of the dehomogenized sieving polynomial $A^{-1}\phi(X,1)$ modulo $p_j \in \mathrm{FB}_{\mathbf{Z}}$ are the data we need in order to be able to perform sieving. Rather than computing the norm form for each of the 2^t inputs to the resultant function we can precompute this norm form with arbitrary indeterminates representing the coefficients of f and coordinates of $\theta - P$ on the standard basis of $\mathbf{K}$. The degree $d = 4$ is small enough that the resulting polynomial has few terms and so it is easy to store and evaluate. A monic defining polynomial can be represented by $X^4 + c_3X^3 + c_2X^2 + c_1X + c_0$ for indeterminates $(c_0, \ldots, c_3)$. The number field elements we consider have the form $Xa_0 + Y\theta - Ya_1$ for indeterminates a_0, a_1. We display the sieving polynomial in dehomogenized form as $a_0^{-1}\phi(X,1)$ as this makes it slightly easier to read (of course $a_0^{-1}\phi(X,Y)$ is what we compute with):

$$a_0^3X^4 - (4a_0^2a_1 + c_3a_0^2)X^3 + (6a_0a_1^2 + 3c_3a_0a_1 + c_2a_0)X^2 - (4a_1^3 + 3c_3a_1^2 + 2c_2a_1 + c_1)X + C \tag{4.11}$$

where $C = a_0^{-1} f(a_1)$.

This approach is different from what Jacobson does in the quadratic case. Recall from Chapter 2 that he uses the fact that quadratic fields are a Galois extension of the rational numbers in order to compute the norm form of the sieving elements A, ω_1. For non-Galois fields such a calculation would not work which is why we employ the multivariable resultant instead.

Now we want to compute the roots of $A^{-1}\phi_i(X, 1)$ modulo each prime in $\mathrm{FB}_{\mathbf{Z}}$. By equation (2.2) we have that

$$\mathrm{N}(XA + \theta - P_i) = \mathrm{Res}_\theta(XA + \theta - P_i, f(\theta)) = f(P_i - AX). \tag{4.12}$$

From this it is clear that $f \pmod p$ for $p \nmid A$ has the same number of roots as $\mathrm{N}(XA + \theta - P_i)$. Fixing a prime $p_j \in \mathrm{FB}_{\mathbf{Z}}$ that does not divide A and a root s_j of f modulo p_j we have

$$\mu_i^{(j)} = \frac{P_i - s_j}{A} \tag{4.13}$$

is a root of $f(P_i - AX) \pmod{p_j}$.

4.4 Self-Initialization

The sieving polynomials $A^{-1}\phi_i(X, Y)$ for $i = 0$ to $2^t - 1$ and the roots $\mu_i^{(j)}$ of the dehomogenized sieving polynomial $A^{-1}\phi_i(X, 1)$ modulo $p_j \in \mathrm{FB}_{\mathbf{Z}}$ are the data we need in order to be able to perform sieving. As we will see below, each ϕ_i is associated with a root P_i of f modulo A. Rather than computing the P_i (resp. $\mu_i^{(j)}$) directly using an algorithm for finding roots over finite fields, our self-initialization approach allows us to order the P_i (resp. $\mu_i^{(j)}$) so that the next value is easily computed from the previous one. In particular, a Gray code ordering is used so that P_i and P_{i+1} (resp. $\mu_i^{(j)}$ and $\mu_{i+1}^{(j)}$) differ at only one prime.

The Chinese remainder theorem allows us to decompose the roots of $f \pmod A$ in terms of the roots of $f \pmod{p_j}$ where p_j are the primes dividing A. We let $\{B_0, \ldots, B_{t-1}\}$ be integers such that

$$B_j \equiv \begin{cases} 0 & \pmod{p_k} \text{ if } k \neq j \\ r_j^{(0)} - r_j^{(1)} & \pmod{p_j} \text{ otherwise} \end{cases} \tag{4.14}$$

and b such that

$$b \equiv r_j^{(1)} \pmod{p_j} \tag{4.15}$$

for all $0 \leq j \leq t$. By the linearity of Chinese remainder theorem we have that

$$b + \sum_{j \in \mathcal{B}} B_j$$

is a root of $f \pmod{A}$ for any set $\mathcal{B}$ that is a (possibly empty) subset of $\{0, \ldots, t-1\}$ and that for two distinct subsets $\mathcal{B}$ and $\mathcal{C}$ these roots are distinct modulo A. Thus the set of values of $b + \sum_{j \in \mathcal{B}} B_j$ over all possible ω_0 is the same as the set of all P. It is convenient to compute B_j using the expression

$$B_j = Ap_j^{-1} \left((Ap_j^{-1})^{-1} (r_j^{(0)} - r_j^{(1)}) \pmod{p_j} \right) \tag{4.16}$$

which follows from Chinese remainder theorem.

We use a Gray code ordering on the solutions $P_0, \ldots, P_{2^t-1}$ so that P_{i+1} and P_i differ at exactly one prime in $\{p_j\}_{j=0}^{t-1}$. In other words, the difference of P_{i+1} and P_i is a multiple of B_k for some k. Each P_i for i from 0 to $2^t - 1$ is associated to a tuple $\mathbf{u} \in \{-1, 1\}^t$ whose ordering is

$$\begin{aligned} \mathbf{u}_0 &= (1, \ldots, 1) \text{ and} \\ \mathbf{u}_{i+1} &= \mathbf{u}_i + (0, \ldots, 0, (-1)^{\lceil (i+1)/2^{\nu_2(2(i+1))} \rceil}, 0, \ldots, 0) \text{ for } 0 \leq i < 2^t. \end{aligned}$$

where the index of the nonzero entry in the tuple added to $\mathbf{u}_i$ is $\nu_2(2(i+1))$. Then the ordering of the P_i is

$$P_0 = b + B_0 + \cdots + B_{t-1} \text{ and} \tag{4.17}$$

$$P_{i+1} = P_i + B_{\nu_2(2(i+1))} \cdot (-1)^{\lceil (i+1)/2^{\nu_2(2(i+1))} \rceil} \text{ for } 0 \leq i < 2^t. \tag{4.18}$$

Next we consider computing the roots by defining an ordering with respect to the index i of the sieving polynomial ϕ_i/A. In particular, let μ_i be a root of this polynomial for a fixed prime $p_j \nmid A$ and root s_j of f modulo p_j. Recall that $\mu_i = \frac{P_i - s_j}{A}$ is a root of $f(P_i - AX) \pmod{p_j}$. The next root is $\mu_{i+1} = \frac{P_{i+1} - s_j}{A}$ which is a root of $f(P_{i+1} - AX) \pmod{p_j}$ for $i < 2^t - 1$. Comparing these expressions for μ_i and μ_{i+1} we see that we can iterate over the roots $\pmod{p_j}$ by

$$\mu_0 = A^{-1}(P_0 - s_j) \text{ and} \tag{4.19}$$

$$\mu_{i+1} = \mu_i + A^{-1}B_{\nu_2(2(i+1))} \cdot (-1)^{\lceil (i+1)/2^{\nu_2(2(i+1))} \rceil} \text{ for } 0 \leq i < 2^t. \tag{4.20}$$

We can compute the roots modulo primes dividing A separately. There are only t such roots so this does not take much time.

4.5 Implementation of Sieving Relation Generation

In this section we specialize to $d = 4$ number fields to be concrete. As we have seen, the high-level details would be unchanged working in an arbitrary fixed degree number field. However, we expect the choices for our implementation would probably need to change as the degree grows.

We can now give the algorithm for computing the sieving data for $[A, \theta - P_j]$ for each j from 0 to $2^t - 1$. Algorithm 1 is the algorithm for generating a sieving polynomial and its roots.

- Λ_j is a tuple of roots of ϕ_j/A which is divided into subtuples of length 0 to 4 depending on the number of roots of f modulo p_i.
- U is a tuple of roots of f which is divided into subtuples of length 0 to 4 depending on the number of roots of f modulo p_i.

Algorithm 1 SievingDataOnePrime

```
Input:
    $j$ : an integer $0 \leq j < 2^t - 1$.
    $P_j$ : a root of $f \pmod{A}$.
    $\Lambda_j$ : a tuple of roots of the $\phi_j/A$.
    $A$ : norm of the ideal being sieved.
    $\mathrm{FB}_{\mathbf{Z}}$ : rational factor base.
    $(B_i)_{i=0}^{t-1}$.
    $U$ : a tuple of roots of the $f$ modulo the primes in $\mathrm{FB}_{\mathbf{Z}}$
Output: $P_{j+1}, \Lambda_{j+1}$
1: if $j = 0$ then
2:     Compute $P_{j+1}$ using (4.17)
3:     Compute roots for $p_i \nmid A$: For each prime $p_i \in \mathrm{FB}_{\mathbf{Z}}$ and each root $(\mu_{j,k})_{k\leq 4} \in \Lambda_j$ of $\phi_j/A$
       modulo $p_i$ compute $\mu_{j+1,k}$ using (4.19) and $U$
4: else
5:     Compute $P_{j+1}$ using (4.18)
6:     Compute roots for $p_i \nmid A$: For each prime $p_i \in \mathrm{FB}_{\mathbf{Z}}$ and each root $(\mu_{j,k})_{k\leq 4} \in \Lambda_j$ of $\phi_j/A$
       modulo $p_i$ compute $\mu_{j+1,k}$ using (4.20)
7: end if
8: Compute $\phi_{j+1}/A$ using (4.11)
9: Compute remaining roots in $\Lambda_{j+1}$ for $p_i \mid A$ using a root finding algorithm
10: Return $P_{j+1}, \Lambda_{j+1}, \phi_{j+1}/A$
```

Algorithm 2 gives the algorithm for generating the sieving data $(P_j, \Lambda_j, \phi_j/A)_{j=0}^{2^t}$ for all the sieving polynomials. The factor base $\mathrm{FB}_{\mathbf{K}}$ is passed as one of the inputs to this algorithm. Prime ideals in $\mathrm{FB}_{\mathbf{K}}$ such that $p \nmid [\mathcal{O}_{\mathbf{K}} : \mathbf{Z}[\theta]]$ are represented in two ways, first by the pair (p, g) where g is a polynomial $\mathbf{Z}[X]$ (which is possible by Theorem 2.6) and, second, by a $\mathbf{Z}$-basis. If p does not divide the index then we store a tuple of pairs of Dedekind-Kummer representations (p, g) for primes in $\mathrm{FB}_{\mathbf{Z}}$ not dividing the index. The other prime ideals in $\mathrm{FB}_{\mathbf{K}}$ are represented by a $\mathbf{Z}$-basis only.

Our algorithm makes use of the roots of f modulo a prime p for $p \nmid [\mathcal{O}_{\mathbf{K}} : \mathbf{Z}[\theta]]$. A root r in this set **corresponds** to the degree one prime ideal $\mathfrak{p} = (p, \theta - r)$ by Theorem 2.6. We set U_0 to be the roots of $f \pmod{p}$ for all $p \in \mathrm{FB}_{\mathbf{Z}}$ such that $p \nmid [\mathcal{O}_{\mathbf{K}} : \mathbf{Z}[\theta]]$. U_0 is straightforward to compute due to the availability of the Dedekind-Kummer representation for prime ideals above p where $p \nmid [\mathcal{O}_{\mathbf{K}} : \mathbf{Z}[\theta]]$. We set U_1 to be the roots of $f \pmod{p}$ for all $p \in \mathrm{FB}_{\mathbf{Z}}$ such that $p \mid [\mathcal{O}_{\mathbf{K}} : \mathbf{Z}[\theta]]$ which we compute using an algorithm for finding roots of polynomials over finite fields. Both U_0 and U_1 are divided into subtuples of length 0 to 4 depending on the number of roots of f modulo p_i. We set U to be the concatenation of U_0 and U_1.

We set $S_{\mathbf{Z}}$ to be the set of rational primes in $\mathrm{FB}_{\mathbf{Z}}$ so that there is a pair of distinct degree one prime ideals $\mathfrak{p}^{(0)}$ and $\mathfrak{p}^{(1)}$ above p and so that $p \nmid [\mathcal{O}_{\mathbf{K}} : \mathbf{Z}[\theta]]$. That is, $S_{\mathbf{Z}} = \{p \in \mathrm{FB}_{\mathbf{Z}} : p \nmid [\mathcal{O}_{\mathbf{K}} : \mathbf{Z}[\theta]], \quad \mathfrak{p}^{(0)}\mathfrak{p}^{(1)} \mid (p), \quad \mathfrak{p}^{(0)} \neq \mathfrak{p}^{(1)}\}$.

The algorithm selects the norm A to use. This should be close to the optimal norm that we computed in Section 4.2. In degree four this simplifies to

$$4^{\frac{-1}{2}}(4-1)^{-1}||f|| = \frac{1}{6}||f||.$$

Initially, we let N be the closest integer to $\frac{1}{6}||f||$ and

$$S_N = (p : p \in S_{\mathbf{Z}}, p \leq N).$$

In the case $|\Delta|$ is small the target norm $\frac{1}{6}||f||$ is very small. It can, of course, be so small that the set of rational primes smaller than it is empty. If there are fewer than 5 primes in S_N we double N until $|S_N| \geq 5$.

A good approach to finding a subset of t primes from S_N so that their product A is close to our target is given in [Jac99b, Algorithm 4.2]. The idea is that we are given t and choose rational primes that are close to $(\frac{1}{6}||f||)^{t^{-1}}$. This ensures that their product is close to $\frac{1}{6}||f||$.

The requirement that the norm A be close to $\frac{1}{6}||f||$ restricts the largest choice of t more than it is restricted in the quadratic case where A should be approximately $\sqrt{|\Delta|}$. We discuss the choice of t in our implementation in Chapter 5.

Algorithm 2 SievingData

```
Input:
    $f$ : a degree four polynomial.
    $|\Delta|$ : discriminant of the number field.
    $[\mathcal{O}_{\mathbf{K}} : \mathbf{Z}[\theta]]$ : index of the defining polynomial.
    $E$ : sieving radius.
    $t$ : an upper bound on the number of rational primes to sieve.
    $\mathrm{FB}_{\mathbf{Z}}$ : rational prime factor base.
    $\mathrm{FB}_{\mathbf{K}}$ : factor base.
Output: SD : sieving data for all $2^t$ polynomials
1: Compute $U_0$ from $\mathrm{FB}_{\mathbf{K}}$ as described above
2: Compute $U_1$: for primes $p \mid [\mathcal{O}_{\mathbf{K}} : \mathbf{Z}[\theta]]$ compute roots of $f \pmod p$ using a root finding algorithm
3: Combine both lists of roots into a sorted tuple $U$
4: Compute $S_{\mathbf{Z}} = \{p \in \mathrm{FB}_{\mathbf{Z}} : p \nmid [\mathcal{O}_{\mathbf{K}} : \mathbf{Z}[\theta]], \quad \mathfrak{p}^{(0)}\mathfrak{p}^{(1)} \mid (p), \quad \mathfrak{p}^{(0)} \neq \mathfrak{p}^{(1)}\}$
5: If $S_{\mathbf{Z}}$ is empty Return "$B$ too small"
6: Store the tuple of two roots $r_j^{(0)}$ and $r_j^{(1)}$ corresponding to two degree one prime ideals above $p$
   for $p \in S_{\mathbf{Z}}$
7: $t = \min\{|S_{\mathbf{Z}}|, t\}$
8: $N = \frac{1}{6}||f||$
9: Set $S_N = (p : p \in S_{\mathbf{Z}}, p \leq N)$ and double $N$ until $|S_N| \geq 5$
10: Find a subset of $t$ primes from $S_N$ so that their product $A$ is close to $\frac{1}{6}||f||$
11: Compute $b \equiv r_j^{(1)} \pmod{p_j}$ for $j$ from 0 to $t-1$
12: Compute $B_j = Ap_j^{-1}((Ap_j^{-1})^{-1}(r_j^{(0)} - r_j^{(1)}) \pmod{p_j})$ for $j$ from 0 to $t-1$
13: for $j$ from 0 to $2^t - 1$ do
14:     $(P_j, \Lambda_j, \phi_j/A) = \mathrm{SievingDataOnePrime}(j, P_j, \Lambda_j, A, \mathrm{FB}_{\mathbf{Z}}, (B_i)_{i=0}^{t-1}, U)$
15:     $\mathrm{SD}_j = (P_j, \Lambda_j, \phi_j/A)$
16: end for
17: Return $\mathrm{SD}, N, t, S_N$
```

We turn to comparing with some prior related work. First we compare with the quadratic case. We have taken what is done in the quadratic case and built on that to get a self-initialization approach that works in number fields of any degree. One way that the method discussed in Section 4.4 differs from Jacobson's self-initialization is in the number of sieving polynomials used for a particular number of rational primes t. We use 2^t sieving polynomials rather than 2^{t-1}. The reason is that for a pair of sieving elements $[A, \theta + P]$ there is usually *not* a corresponding pair $[A, \theta - P]$ as there is in the case when the defining polynomial is of the form $f = X^2 - D$ for some $D \in \mathbf{Z}$. For example, suppose the sieving norm is a single prime p. Then we want to sieve the two degree one prime ideals

$$P_j^{(0)} = (p, r_j^{(0)}) \text{ and } P_j^{(1)} = (p, r_j^{(1)})$$

we have chosen above p. When the defining polynomial is $X^2 - D$ then $r_j^{(0)} = -r_j^{(1)}$ holds for every p. Unfortunately, for general defining polynomials this rarely happens (the probability it happens for a particular prime is $1/p$). For this reason we cannot take advantage of symmetry in the way that Jacobson does in order to obtain the same number of relations from half the number of sieving polynomials.

We examine the distinctions between our method and Biasse-Fieker's approach as presented in [BF12], previously discussed in Section 3.6. Their work is relevant for two reasons: firstly, for its use of sieving in relation generation, and secondly, as it is the most recent relation generation approach documented in the literature for number fields of degree larger than two. A major difference is that unlike the approach of Biasse-Fieker, our approach can be applied to any number field in the sense that it is not dependent on finding elements of small height within the number field. Finding elements of small height is a generalization of the problem of finding a defining polynomial of small height. As stated in Section 3.6, we are not aware of any algorithm, even one that is exponential time, that finds a defining polynomial of height upper bounded by $\sqrt{|\Delta|}$.

Another difference is that Biasse-Fieker derive $|\,\mathrm{FB}_{\mathbf{K}}\,|$ relations from sieving a single ideal $\mathfrak{a} = (1)$. The Biasse-Fieker approach described in [BF12, Algorithm 3–4] sieves $\mathfrak{a} = (1)$ to get $|\,\mathrm{FB}_{\mathbf{K}}\,|$ relations. They say to do this by repeating the algorithm until a large number of relations from (1) is found. This is done by generating different randomization vectors $\mathbf{a}$. Biasse-Fieker do not give details on how $\mathbf{a}$ should be chosen.

Our self-initialization approach sieves 2^t distinct ideals generated by $(A, \theta - P_i)$ for i from 0 to $2^t - 1$. As we do not do an LLL reduction we can be fully explicit about the elements being tested for smoothness. We test the elements in the sets

$$T_i = \{xA + y\theta - yP \ : |x| \le E, |y| \le E\}$$

for smoothness.

In assessing efficiency, we first consider the time required by our method to initialize the pair of sieving elements $[A, \theta - P_i]$ which follow $[A, \theta - P_{i-1}]$ in the iteration of sieving elements. In other words, to compute P_i given P_{i-1}. The Biasse-Fieker approach must compute a pair of elements $[\alpha, \beta]$ using LLL. This is relatively time-consuming compared to the self-initialization approach which only requires a small number of multiplications and additions of integers and so is much faster than LLL.

Next, we consider computing sieving polynomials. The method Biasse-Fieker use to compute the norm form is interpolation (see [Nei02] for details) for each new pair of elements $[\alpha, \beta]$ The interpolation approach must be reused for each pair of sieving elements and so becomes expensive as the number of sieving polynomials grows. Our approach is to compute a general norm form that can be used for any pair of sieving elements $[A, \theta - P]$. The degree four sieving polynomial's five coefficients are given by multivariable polynomials in equation (4.11). Each of them has low degree and a small number of terms and so evaluating these polynomials is much faster than computing the norm form by interpolation.

Finally, in their approach no method of computing the roots of the sieving polynomials is suggested and so it appears necessary to resort to using a root finding algorithm over a finite field to obtain the roots. In our approach the roots of the sieving polynomial ϕ_i/A are computed from the roots of ϕ_{i-1}/A which is done with multiplication and addition of integers and so is much faster than using a root finding algorithm.

Chapter 5

Implementation Description

Contents

In Chapter 4 we described Algorithm 2, a self-initialization relation generation algorithm. This algorithm can be applied to quartic number fields, and we focus exclusively on these fields in the sequel. This chapter presents a complete index-calculus implementation using Algorithm 2 as the primary method for generating relations. This implementation [Mar24] is composed of 10,000 lines of our own code as well as leveraging subalgorithms in the libraries `PARI/GP`, `Magma`, `Flint`, and `CADO-NFS`. Our implementation is independent of other class group algorithm implementations and our code incorporates several optimizations like a large prime variation. The standalone design of our implementation makes it easy to modify and extend. The decision to develop a new implementation was intended to make benchmarking straightforward and to avoid the constraints inherent in existing software libraries.

The software libraries we used and hardware we tested on are discussed in Section 5.1. Our implementation is discussed in Section 5.2. In Section 5.2.5 a pseudocode outline of the implementation is given. In Section 5.3 we discuss how the parameters of the implementation were chosen.

5.1 Software and Hardware

We use the following software libraries:

- `CADO-NFS`: v2.3.0 [Tea23a]
- `Flint`: v2.8.4 [tea23b]
- `Magma`: v2.27-6 [CBFS23a] [BCP97]
- `PARI/GP`: v2.11.2 [The20]
- `SageMath`: v10.0 [The23]

The `CADO-NFS` project can be seen as having two pieces. First, it contains a collection of programs which can be run in a standalone manner and together form a pipeline for a complete index-calculus algorithm. Second, it contains software for executing this entire pipeline on a large number of machines in a semi-automated way.

We have used the sieving program `las` from `CADO-NFS` in our relation generation stage. We chose this program because it is a very mature implementation and incorporates many optimizations. Although it began as an implementation of the lattice sieving algorithm in [FK05], over time a lot of additional functionality has been added. One advantage of using `las` for sieving is its support for large prime variations. Some further details on `las` are given in [KAF$^+$10, Appendix A]. The lecture notes of [Tho22] cover `CADO-NFS` in detail and include information about `las` not found in `CADO-NFS`'s documentation.

`Flint` is a C library for doing number theory that provides arithmetic in rings including the integers, rationals, reals, and number fields. We use it for some computations with dense matrices over the integers and real numbers. Although doing all linear algebra in `Magma` would be more optimal, `Flint` was originally chosen because of features like support for algebraic numbers and ball arithmetic over the reals. These features could be used in the future.

`Magma` is a well-supported software package that includes many algorithms for algebra and number theory amongst other areas. `Magma` includes a high-level interface to make it easier for users to make use of its algorithms which are written in low-level code for performance. Unlike the other software packages we use, `Magma` is paid software. Its most important algorithms are proprietary and cannot even be viewed by `Magma` license holders. We use it for its state-of-the-art linear algebra algorithms for integer matrices which were already discussed in Section 3.1.5.

`PARI/GP` is a computer algebra system designed for fast computations in number theory. Like `Magma`, `PARI/GP` includes a high-level interface, `GP`, to make it easier for users to use its performance algorithms. We use it for its number field algorithms.

`SageMath` is a mathematics software system that provides a high-level `Python` interface to many existing software packages including `Magma`, `PARI/GP`, and `Flint`. Like [BF12], we use the `SageMath` library to interface between the other libraries in order to access their high-performance algorithms. By doing the computationally expensive parts of the implementation with these subalgorithms, the overall implementation can be kept fast. Using an interface does mean some overhead but the amount of overhead is primarily a function of the number of calls that are made to a particular function. Fortunately, the functions available in these libraries are expressive enough that we only need to call the most time-consuming ones a small number of times (fewer than ten times) in the most important cases. The cases where we need to call the same function thousands of times represent less than 1% of the total running time.

One advantage of using `SageMath` is that it includes functionality from many different libraries. Functions in these libraries can be used in a modular manner. We can swap out one implementation for an alternative one with ease. The large library of available functions enabled us to iterate through several approaches to linear algebra (discussed in Section 5.2.4). Using this approach has made it easy to structure our project in a way that makes it modifiable in the future. Another advantage is the performance of conversion between data types. For example, the implementation needs to convert a dense matrix in `Flint` representation to a matrix in `Magma`'s sparse representation. Sage's conversions between the data types from `PARI/GP`, `Magma` and `Flint` is efficient enough for our purposes.

As `SageMath` already integrates with `Magma`, `PARI/GP`, and `Flint` and the only code we needed to add to interact with other software was for `las`.

All the timings reported in Chapter 6 were done on a server at University of Calgary consisting of Intel Xeon E7 CPUs with 2.80GHz clock speed and 32 KiB L1 cache.

We also did some testing of the sieving stage on an Apple M1 Pro CPU with 128 KiB L1 cache. On this system, the time for `las` was roughly 6 times faster for the same set of parameters. This is interesting as `las` is presumably most optimized for x86 processors and not ARM-based processors like the M1. We expect an x86 machine with similar performance characteristics to the M1 should be even faster. This illustrates the difference that hardware choices can make. It is also a reminder that the range of discriminants that are feasible to compute with our implementation could be extended by splitting the various steps of the index-calculus algorithm so that each step is run on the available hardware that is best-suited for it.

5.2 Implementation

In this section we discuss our implementation in terms of four phases: initialization, relation generation, relation processing, and linear algebra. These parallel the four phases discussed in the chapter on prior work in Section 3.1.

5.2.1 Stage 1 Initialization

We are given a defining polynomial f for a quartic field $\mathbf{K}$ and the absolute value of its discriminant $|\Delta|$. We must compute some invariants and properties of it that are used in the rest of the algorithm. In Section 4.1 we explained why it can be assumed that we have a defining polynomial that is λ-reduced with $\lambda \leq \frac{1}{6}$. So we will assume our defining polynomial f is given in this form.

In Section 3.1.1 we discussed the importance of choosing the smoothness bound B for the efficiency of the self-initialization implementation. We express B as a function of the best bound on a set of prime ideals generating the ideal class group. Recall from Section 2.6 that currently the best bound on a set of prime ideals generating the ideal class group is the bound of Belabas-Diaz-Friedman and we denote it by $T_{\min}$. How we chose B is discussed in Section 5.3.

At present, B is selected this way:

- compute $T_{\min}$ with `GRHBound`($\mathbf{K}$).
- set $B = T_{\min}$ for $|\Delta|$ with fewer than 100 bits, and $B = 1.5 \cdot T_{\min}$ otherwise.

The choice of $B \geq T_{\min}$ is also used by `Magma` although `Magma`'s values are usually larger. `Magma` sets the smoothness bound to always be at least as large as $T_{\min}$ but as $|\Delta|$ grows B is chosen larger than $T_{\min}$. As a representative example of `Magma`'s B selection we generated a random quartic field with 208 bit discriminant. We found $T_{\min} = 25061$ and $B = 59605$, about 2.4 times larger than $T_{\min}$. See Chapter 6 for tables that include the B used by `Magma`.

Our approach differs from Jacobson's strategy in SIQS. As discussed in Section 3.1.8, he finds that the total running time could be improved by setting B *smaller* than the best bound at the time, Bach's. Two differences between our implementation and Jacobson's are relevant:

- $T_{\min}$ is much smaller than the Bach bound,
- the state-of-the-art for linear algebra over $\mathbf{Z}$ has advanced.

The first point has been covered earlier. Regarding the second, more efficient algorithms for computing the HNF and kernel allow for managing a larger B. Consequently, it is unsurprising that the optimal B would be greater than $T_{\min}$.

We now turn to the initialization data that we will need in later stages of the algorithm. The descriptions in Chapter 2 of these invariants and properties can be referred to for further details.

To compute which rational primes can be used for sieving we require the number field's index $[\mathcal{O}_{\mathbf{K}} : \mathbf{Z}[\theta]]$. Our implementation computes the index using `PARI/GP`'s `nfinit` function.

An integral basis of $\mathcal{O}_{\mathbf{K}}$ is needed in order to factor prime ideals and compute valuations. For degree four number fields, computing an integral basis requires the Round 2 algorithm and the prime factorization of $|\Delta|$. Our implementation computes an integral basis of $\mathcal{O}_{\mathbf{K}}$ using `PARI/GP`'s `nfinit` function.

The signature of the number field is, of course, necessary and our implementation also obtains this from `nfinit`.

In order to compute the factor base we must compute the prime ideals above p. When p divides the index this requires an integral basis of $\mathcal{O}_{\mathbf{K}}$ and is done with a sophisticated algorithm described in [Coh93, Algorithm 6.2.9] and available in `PARI/GP` using the `idealprimedec` function. Our implementation computes $\mathrm{FB}_{\mathbf{K}}$ in the following way:

- For each $p \leq B$ we compute the primes $\mathfrak{p}$ above p using `PARI/GP`'s `idealprimedec` function.
- We then compute $N(\mathfrak{p})$ and discard those $\mathfrak{p}$ so that $N(\mathfrak{p}) \geq B$. The result is stored as $\mathrm{FB}_{\mathbf{K}}$.
- `PARI/GP` represents prime ideals using a $\mathbf{Z}$-basis and in a two-element representation. If p does not divide the index then the two-element representation is given by Theorem 2.6 (Dedekind-Kummer). We store a tuple of pairs of Dedekind-Kummer representations (p, g) for primes in $\mathrm{FB}_{\mathbf{Z}}$ not dividing the index.

The roots of unity in any number field form a cyclic group $\langle \omega_m \rangle$ of order m where $m \mid 4$. Our implementation calls `nfrootsof1` from `PARI/GP` to compute m, ω_m.

Our implementation computes an approximation $\widetilde{h}\widetilde{R}$ to hR. We use the approach to approximating to hR using Euler products that is described in [Coh93, Algorithm 6.5.6]. We implemented this approach. The prime ideals involved in the product are implemented in `PARI/GP`.

After these initial computations are done we need to prepare the data for self-initialization. We call Algorithm 2 which returns SD, t. SD consists of triples of the data needed for sieving ϕ_j/A which are $(P_j, \Lambda_j, \phi_j/A)$ and t is the number of sieving primes that were used.

5.2.2 Stage 2 Sieving

Sieving is done by executing the `las` binary from `CADO-NFS`.

Two important parameters of `las` are the `fb` and `poly` parameters. To construct these parameters we loop over SD. Recall that its entries consist of a triple with the following components:

- the integer P_j used to define the sieving pair
- $[A, P_j]$, the roots of the sieving polynomial modulo the rational primes in the factor base Λ_j
- and the sieving polynomial ϕ_j/A

We convert the Λ_j and ϕ_j/A to the format used by `CADO-NFS`. The parameter `fb` of `las` is used to pass in the converted Λ_j and the parameter `poly` of `las` is used to pass in the converted ϕ_j/A.

The parameters E and B are passed in to the self-initialization implementation by the user. The parameter `lim` of `las` is used to pass in the smoothness bound. We set `lim` $= B$. The parameter `I` of `las` is used to control the area that is sieved. We set `I`$= \lfloor \log_2(E) \rfloor$.

Recall from Section 3.6 that [BF12] made use of special-q to improve the efficiency of relation generation. `las` supports special-q sieving and we took advantage of this feature. We believed special-q would be a less valuable optimization than it is for `CADO-NFS` because we are already sieving with multiple polynomials, whereas in Number Field Sieve only one sieving polynomial is selected, to which special-q adds some of the benefits of sieving multiple polynomials. Nevertheless, we found a special-q interval of size 50 starting with the smallest prime that factors over $\mathrm{FB}_{\mathbf{K}}$ benefits the total running time. Note that this range is much smaller than what was used by Biasse-Fieker [BF12] which used special-q all the way up to B. The selection of this parameter is discussed in Section 5.3.

The `las` program supports relation generation with as many as eight large primes. It is well known (see [BJ10]) that the majority of the benefit of a large prime variation comes from the first prime with diminishing returns for each additional prime. For this reason we chose to implement a single large prime variation. As with the small prime smoothness bound, `las` sets its large prime bound as a power of two. The exponent is specified by the parameter `lpb`. In the course of writing our implementation, we found that setting the large prime bound LB as a function of $|\Delta|$ worked well. We use the exponent `lpb` $= \lfloor 0.1 \cdot \log_2(|\Delta|) \rfloor$ thus making the large prime bound $LB = \lfloor 0.1 \cdot \log_2(|\Delta|) \rfloor + 5$. The selection of this parameter is discussed in Section 5.3.

As `las` is customized to the integer factorization problem, the relations it generates have two “sides”, an algebraic side and a rational side. We only need to sieve a single side but `las` requires us to pass parameters for both sides. We accommodate it by passing the parameters listed above to both sides. Any element of B-smooth norm found on the first side is also B-smooth on the second side. Although this is inelegant, we note that this does not imply a doubling of the time required for sieving as `las` is optimized to filter the elements examined on the second side so that only the "survivors" from the first side are examined (see the notes on sieving in [Tho22]). The approach causes some overhead, but it is much less than a factor of two.

5.2.3 Stage 3 Relation Processing

In this phase the relations are read, large primes are dealt with, and other operations are performed to make the dimensions of the resulting relation matrix smaller.

Use of the special-q sieving introduces a large number of duplicates which we now remove. For the relation file corresponding to each initial ideal the duplicate relations found by lattice sieving are removed by calling the UNIX commands `sort` and `uniq`.

Our implementation then reads relations from the relations files that `las` outputs.

In Section 3.1.4 we discussed how an implementation using a large prime variation must load a larger number of partial relations than an implementation without this variation in order for the algorithm to obtain the same final result. This observation comes from previous work on large prime variations such as Jacobson-Biasse [BJ10] and [BKV19].

In the course of writing our implementation, we found that our implementation often performed well when parameters are selected so that roughly $k_{\text{req}} = 3|\,\text{FB}_{\mathbf{K}}\,|$ partial relations are found. At present, our implementation's parameters have been selected with the aim of collecting $3|\,\text{FB}_{\mathbf{K}}\,|$ partial relations.

Often k_{req} will be smaller than the number of relations found by sieving the 2^t polynomials. In this case we will use k_{req} of the relations and keep the additional relations in reserve. If we later find that more relations are needed then we load 20 more. These relations are selected in the following manner. We ensure that a similar proportion of relations are loaded for each initial ideal $\mathfrak{a}$ by choosing a random subset of size $\lfloor k_{\text{req}}/2^t \rfloor$ that will be loaded.

In contrast, Jacobson [Jac99b] loads relations polynomial-by-polynomial. We chose this approach based on [BF12]'s statement that a difficulty with their sieving approach is that it produces sets of relations with more linear dependencies than other approaches. By loading a roughly equal proportion of relations for each sieving polynomial we hope to make the loaded relations have slightly fewer linear dependencies than if we loaded them polynomial-by-polynomial. We note that since we did not implement the alternative approach there is no easy way to demonstrate that a smaller number of linear dependencies is achieved.

We then zero-initialize (initialize all entries with the value 0) the `Flint` matrices M and $M_{\log}$ where M is a dense integer matrix, $M_{\log}$ is a dense matrix with real numbers implemented using the arbitrary precision real number library `mpfr`. Both have k_{req} rows and they have $|\,\text{FB}_{\mathbf{K}}\,|$ and $r_1 + r_2 - 1$ columns respectively. For large primes we zero-initialize lp, which identifies the rational primes dividing the norm of a partial relation, and Lp which holds the valuations at the prime ideals above p. These four objects specify our partial relations.

Element Arithmetic

Next we loop over the sieving data SD which is a tuple of triples $P_j, \Lambda_j, \phi_j/A$. Thus we have the pair $[A, \theta - P]$ of ideal elements that were used to define the sieving polynomial ϕ_j/A and the file of stored relations for this polynomial.

The lines in the relations files consist of a pair of integers x, y and the norm $\mathrm{N}(xA + y\theta - yP)$. We need to change these lines into relations that can be inserted into the matrices M and $M_{\log}$. To do this we loop over the lines in the files. We let:

- i be the index of the relation we are processing,
- $\tau = x_i A + y_i \theta - y_i P_j$ be the element of the current relation,

Each new element is added to a set PE. The set PE is represented as a `Python` set of `PARI/GP` number field elements. A set is used because PE is only used to check what elements have been encountered. The set type is used because the underlying data structure is a hash table and thus set membership has expected time complexity $O(1)$. Consequently, this check is fast.

Futher Deduplication

Our previous deduplication eliminated relations defined on elements τ and τ' which had the same coordinates with respect to the sieving elements $[A, \theta - P]$.

We now check if we have encountered the current relation before among the whole set of relations. This means that we check if we have previously encountered an element $x'A + y'\theta - y'P$ that is equal to τ up to multiplication by a root of unity. In the next deduplication we will also check for pairs of relations that are multiples of non-torsion units. In practice this is extremely rare so we only do it once. If we have such an equality the corresponding vectors in the relations lattice are the same. Consequently, adding this relation would be useless.

From our calculation in the initialization stage, we have the order m of the group of roots of unity and a generator ω_m of this group. We check if $\tau \cdot \omega_m^i \in \mathrm{PE}$ for $i = 0, \ldots, m-1$. If it is, the current relation is discarded and we continue.

Identify Large Prime

Next, we determine if a large prime is present. Our implementation checks if there is a rational prime p in the list of factors of $\mathrm{N}(\tau)$ such that p is larger than B. If there is one we set $\mathrm{lp}_i = p$. We call `idealPrimeDec` to find the degree one prime ideals above p and temporarily stores them in a tuple $\mathcal{P} = (\mathfrak{p}_i)_{i=0}^{l}$ where l is between 1 and 4.

Valuations

Up to this point have only considered τ, an element of B-smooth norm. To determine a partial relation from it we need to compute the valuation of τ at the prime ideals above the rational primes dividing $\mathrm{N}(\tau)$. Recall an element of B-smooth norm is not necessarily $\mathrm{FB}_{\mathbf{K}}$-smooth (due to the existence of prime ideals of degree larger than one) so not every τ will give us a partial relation. In the course of writing our implementation, we found that most of them will.

In the quadratic case the valuation of an element α at a prime ideal $\mathfrak{p}$ above a rational prime $p \mid \mathrm{N}(\alpha)$ can be computed efficiently via the Kronecker symbol of $\mathrm{N}(\alpha)$ modulo p (see [Jac99b, Theorem 2.4]). In the case of a degree four number field, valuations are more complicated. The state-of-the-art is [Bel04, Algorithm 5.11] which is implemented in `PARI/GP` as the function `valuation`.

Our implementation computes $\nu_{\mathfrak{p}}(\tau)$ for each $\mathfrak{p}$ in the factor base and for $\mathfrak{p}$ above the large primes. If $\mathrm{lp}_i \neq 0$ it does the following:

- For each small prime p compute the valuations $v = \nu_{\mathfrak{p}}(\tau)$ for each $\mathfrak{p} \in \mathrm{FB}_{\mathbf{K}}$ that is above p,
- If there is a large prime p we compute $v = \nu_{\mathfrak{p}}(\tau)$ for each prime ideal $\mathfrak{p} \in \mathcal{P}$ (recalling that $\mathcal{P}$ is the set of prime ideals above p),
- Check that candidate relations are true relations by checking if $\prod \mathfrak{p}_i^{e_i} = (xA + y\theta - yP)$, where the product runs over the prime ideals found in the previous steps,
- If the candidate relation is not a true relation, continue,
- Otherwise, update row i in M and Lp at the entries corresponding to nonzero values of v,

In the course of writing our implementation, we found that we investigated whether a custom implementation of the valuation algorithm [Bel04, Algorithm 5.11] would make computing valuations faster. This involved implementing a version of Algorithm 5.11 using `Flint`'s algebraic number class. However, tests suggested the cost of converting the $\mathbf{Z}$-bases of prime ideals into `Flint` would probably outweigh the benefit of a faster valuation algorithm so other, more valuable, optimizations were pursued.

Combining Partial Relations

The matrices $M, M_{\log}, \mathrm{Lp}$ and the tuple lp are the representation of our partial relations. We now have found k_{req} relations.

We now describe how partial relations are combined into full relations. Consider the partial relations that are divisible by the prime $\mathfrak{p}$. From a partial relations

$$(\mathbf{v}_1, \operatorname{Log}\gamma_1, \mathfrak{p}), \ldots, (\mathbf{v}_a, \operatorname{Log}\gamma_a, \mathfrak{p})$$

there are $\binom{a}{2} = a \cdot (a-1)/2$ possible quotients (γ_i/γ_j) for $i \neq j$ each of which factors over $\mathrm{FB}_{\mathbf{K}}$ and so gives us a full relation.

In Section 3.1.4 we discussed strategies used for combining partial relations into full relations. The following strategy is intended to be as simple as possible while still meeting the three requirements from that section. Among the a partial relations we find the index j of the partial relation so that $\mathbf{v}_j$ has the fewest nonzero entries. Then (γ_i/γ_j) gives us a full relation for any $\gamma_i \neq \gamma_j$. We select these $a-1$ relations as our full relations.

The first requirement is that the set of full relations we construct spans all the full relations that we could possibly derive. In other words we want full relations that span the $a \cdot (a-1)/2$ relations coming from the quotients γ_i/γ_j. Indeed, given an arbitrary $\gamma_i/\gamma_{i'}$ we can write $\gamma_i/\gamma_{i'}$ as

$$\frac{\gamma_i}{\gamma_{i'}} = \frac{\gamma_i}{\gamma_j}\left(\frac{\gamma_{i'}}{\gamma_j}\right)^{-1}$$

so the first requirement is satisfied.

Example 5.1. *As one example, with a set of two partial relations corresponding to field elements* γ_1, γ_2 *we have* $2-1 = \binom{2}{1}$, *and we have a single full relation corresponding to the element* γ_1/γ_2. *As another example, a set of three partial relations corresponding to elements* $\gamma_1, \gamma_2, \gamma_3$ *yields three possible full relations corresponding to the elements* $\gamma_1/\gamma_2, \gamma_2/\gamma_3, \gamma_3/\gamma_1$. *Each of these quotients corresponds to a difference of two partial relations, and we see that subtracting the first relation from the second relation gives the third relation.*

The second requirement is that a minimal number of linear dependencies are introduced into the full relations. As [Cav02, Section 3.3.2] indicates, any subset of $a-1$ out of the total $a \cdot (a-1)/2$ are linearly independent as abstract vectors. Thus our strategy does not introduce any avoidable linear dependencies into the full relations.

The final requirement is that the increase in density and increase in the size of coefficients of the relations matrix is minimized. Our approach is certainly not as good as Biasse-Jacobson's [BJ10] approach which is currently the state-of-the-art technique in the context of computing class groups. However, as the number of partial relations divisible by a given large prime is relatively small compared to the total number of relations found, their more sophisticated approach would not be that beneficial.

The corresponding full relation is found by subtracting row j from row i in M and $M_{\log}$. After performing the $a-1$ subtractions, the row at index j is the only one still corresponding to a relation divisible by $\mathfrak{p}$ and can be discarded.

Our implementation does the following for each $\mathfrak{p} \in \text{lp}$:

- Among the partial relations divisible by $\mathfrak{p}$ compute the index j of the row in M so that $\mathbf{v}_j$ has the smallest weight.
- Iterate over the indices i of rows divisible by $\mathfrak{p}$ not including $i = j$ and subtract row j from row i in $M, M_{\log}$
- Remove row j from M and $M_{\log}$

At the end of this process the large prime data in Lp and lp is no longer needed and is discarded.

We note that combining partial relations is not the only structured Gaussian elimination that is performed. Further structured Gaussian elimination is done by our linear algebra approach (see Section 5.2.4). This is analogous to the approach of Jacobson in [Jac99b] where structured Gaussian elimination was done by an elimination approach of [HM97] in the course of computing the HNF.

Final Deduplication

We now have a set of full relations. Some of these may be duplicates and it is worthwhile to eliminate them. We do this by fixing a row $\mathbf{r}_i$ of M and iterating through the rows to find duplicates. For any two duplicate rows $\mathbf{r}_i$ and $\mathbf{r}_j$ of M we know that γ_i/γ_j is a unit. It could be either a root of unit (torsion unit) or a non-torsion unit. In the course of writing our implementation, we found that we never encountered a non-torsion unit but if we did such a unit could be stored and used later when we attempt to compute the unit group. For this reason our implementation checks for it before eliminating one of the rows. This is done by testing if $\gamma_i^m = \gamma_j^m$ where m is the integer so that $\langle \omega_m \rangle$ is the group of roots of unity. If this holds then γ_i and γ_j differ by a root of unity and so the second relation can be discarded.

Trivial Relations

The implementation iterates over the primes p in $\mathrm{FB}_{\mathbf{Z}}$ and identifies the ones so that all the prime ideals $\mathfrak{p}$ satisfy $\mathrm{N}(\mathfrak{p}) < B$. Put another way, we are identifying the primes p so that (p) is a B-smooth ideal (recall that for degree larger than 2 a principal ideal whose norm is B-smooth is not necessarily a B-smooth ideal because of the existence of prime ideals of degree larger than one). Let $\prod \mathfrak{p}_i^{e_i}$ be the factorization of such a prime and $\mathbf{e}$ be the tuple of the exponents. Then $(\mathbf{e}, \operatorname{Log} p)$ is a relation. We append $\mathbf{e}$ to M and $\operatorname{Log} p$ to $M_{\log}$.

The number of such relations is fairly small but they play an important role in structured Gaussian elimination because they have low weight and their nonzero entries are in the columns of the matrix that are most dense.

Additional Relations

The goal of relation generation is to obtain enough full relations so that

$$\widetilde{hR} < \widetilde{h}\widetilde{R} < 2 \cdot \widetilde{hR}.$$

Recall that $\widetilde{h}$ is defined as 0 if the lattice generated by the vectors $\mathbf{v}$ in our relations $(\mathbf{v}, \operatorname{Log} \alpha)$ does not have full rank. Thus we must find at least $|\,\mathrm{FB}_{\mathbf{K}}\,|$ relations for equation (3.1) to be satisfied. In practice, more than $|\,\mathrm{FB}_{\mathbf{K}}\,|$ are needed.

As discussed in Section 3.1.7 a standard approach has been to economize on the number of relations passed to the HNF as much as possible. Recall that in this approach we stop generating full relations after we have $|\,\mathrm{FB}_{\mathbf{K}}\,| + c$ of them for a small $c > 0$. After generating the initial $|\,\mathrm{FB}_{\mathbf{K}}\,| + c$ relations we proceed to some approach to linear algebra, and based on this approach, find additional relations.

We also described that when structured Gaussian elimination is used choosing a number of relations that is $a|\,\mathrm{FB}_{\mathbf{K}}\,|$ for a real number $a > 1$ can be helpful. In the course of writing our implementation, we found that using $a|\,\mathrm{FB}_{\mathbf{K}}\,|$ for $a > 1$ was also useful for increasing the rank of M.

A larger rank means that less additional relation generation is needed in order to make M have full rank. At present, our implementation requires $1.2|\operatorname{FB}_{\mathbf{K}}|$ full relations be generated before continuing to the linear algebra stage. After this number of relations is collected we compute the rank of M. We do this by computing the linearly dependent columns in M using the `Flint` function `fmpz_mat_rank`.

If the rank is smaller than $|\operatorname{FB}_{\mathbf{K}}|$ we set $\widetilde{h} = 0$. In this case, we need to generate additional relations. As discussed in Section 3.1.7 sieving relation generation can be used to construct relations that are targeted to the missing primes. This is a well-known approach described in [Jac99b, Algorithm 4.3], for example. In Algorithm 4.3 Jacobson seems to indicate he used the same parameters (sieving radius, smoothness bound, etc.) for his targeted approach as were used for the initial relation generation. In the course of writing our implementation, we found that using a sieve radius of E for the targeted sieving often yielded no relations. Our implementation uses a sieving radius of $4 \cdot E$ as it was the smallest radius we tested that yields at least one relation. All other parameters are the same except we do not do a large prime variation for our targeted approach. A large prime variation is yet another optimization that future work could experiment with.

5.2.4 Stage 4 Linear Algebra

As discussed in the chapter on prior work, Chapter 3.1.5, linear algebra is a critical stage in terms of the efficiency of the algorithm. In the initial version of our implementation we used `Flint`'s algorithms for linear algebra. This made the linear algebra stage the computational bottleneck. Therefore multiple approaches were implemented before we selected the final approach. We briefly describe the three approaches we experimented with. Since the goal of these experiments was to choose a final approach, not to develop finalized versions of multiple approaches, we will not give timings for the variants that we did not end up selecting.

In testing these ideas we used $|\Delta|$ of size between 100 and 150 bits. The lengths of the factor base were between 250 and 1500. The bottleneck of the linear algebra is always the computation with M as it has $|\operatorname{FB}_{\mathbf{K}}|$ rows and $O(|\operatorname{FB}_{\mathbf{K}}|)$ columns.

The first approach we implemented was the well-known approach of computing the elementary divisors by computing the HNF and computing the regulator using the transformation matrix of the HNF. This was tried both with `Flint`'s HNF (`fmpz_mat_hnf_transform`) and `Magma`'s HNF (`HermiteForm`). `Flint`'s implementation of the HNF is based on the paper [PS10] which in turn makes use of (unpublished) work of Allan Steel on `Magma`'s HNF. Our comparisons showed `Magma`'s

HNF is enormously faster than `Flint`'s.

The second approach we implemented was to obtain the class group by computing the HNF without the transformation matrix and compute the unit group using Vollmer's approach. Vollmer's approach was mentioned in Section 3.1.5 in the context of computing the regulator of a real quadratic field. With a small modification, this can be applied to the problem of computing the regulator of a general number field. This modification is as follows: Given M and a relation $(\mathbf{r}, \log \alpha)$ we solve $\mathbf{X}M = \mathbf{r}$ for $\mathbf{X}$ using the `Flint` function `fmpz_mat_solve`. We augment M to $M = \left[\begin{array}{c} M \\ \hline \mathbf{r} \end{array}\right]$. Then we have $\left[\begin{array}{c|c} \mathbf{x} & -1 \end{array}\right]$ is in the nullspace of the new M. Thus $\left[\begin{array}{c|c} \mathbf{x} & -1 \end{array}\right] \cdot \left[\begin{array}{c} M_{\log} \\ \hline \operatorname{Log} \alpha \end{array}\right]$ is a vector in the log unit lattice. We iteratively construct $\left[\begin{array}{c|c} \mathbf{x} & -1 \end{array}\right]$ this way and then augment M with them. After repeating this some number of times we check if we have obtained a basis for the log unit lattice. The advantage of Vollmer's method is that computing the HNF without the transformation matrix and solving the systems of equations is much faster than computing the HNF with the transformation matrix. This implementation was done using `Flint` integer matrices. Solving the linear systems that arise in Vollmer's approach was also done in `Flint`.

The third approach we implemented computes the elementary divisors using the `Magma` sparse integer matrix function `ElementaryDivisors`. It computes the unit group by obtaining a basis of M's kernel with the `Magma` sparse integer matrix function `KernelMatrix`. According to `Magma`'s documentation, the algorithm first performs structured Gaussian elimination using Markowitz pivoting and then calls `Magma`'s kernel algorithm for dense matrices. In the course of writing our implementation, we found that function `ElementaryDivisors` is very roughly one hundred times faster than computing elementary divisors using `Magma`'s HNF. The function `KernelMatrix` is very roughly ten times faster than computing the `Magma`'s HNF along with the transformation matrix.

Our implementation uses the third approach because it is faster than the others were. As a first step we convert M to a sparse `Magma` matrix. We then call the functions `ElementaryDivisors` and `KernelMatrix` on this matrix. The switch to `Magma`'s linear algebra shifted the bottleneck in the overall algorithm's running time from linear algebra to relation generation.

We now have a tuple of elementary divisors returned by `Magma` and a `Magma` dense integer matrix holding the kernel. We obtain $\widetilde{h}$ as the product of the elementary divisors. To complete computation of the regulator, the kernel matrix is converted from a dense integer matrix in `Magma` to `Flint`. To compute the regulator and fundamental unit matrix we followed the standard approach used in [Coh93, Algorithm 6.5.7]. Our implementation of Algorithm 6.5.7 uses the real logarithmic embedding rather than Cohen's complex logarithmic embedding. We use an initial precision of $\texttt{prec} = 2^9$. If the precision is too low then Algorithm 6.5.7 can give results that are not numerically accurate. In particular, the `rgcd` function will return a result that is smaller than the true value and, consequently, $\widetilde{h}\widetilde{R}$ will never fall into the range $\widetilde{hR} < \widetilde{h}\widetilde{R} < 2 \cdot \widetilde{hR}$. If we observe this, we follow the standard technique of dealing with precision issues by restarting the algorithm with a higher precision [BF12].

At this point we have a nonzero candidate class number $\widetilde{h}$ and a candidate regulator $\widetilde{R}$. We recall that $\widetilde{hR}$ is the approximation of the product of the class number and regulator that we have computed. To know our lattice of relations is not a sublattice we need $\widetilde{hR} < \widetilde{h}\widetilde{R} < 2 \cdot \widetilde{hR}$ to hold.

At this point it is very likely that one, or even both, of $\widetilde{h}$ and $\widetilde{R}$ are small multiples of their true values. Biasse-Fieker [BF12] also observed this behavior in their sieving algorithm which they tested on number fields of degree up to 6. They also note that

> "it frequently even looks like only a factor of 2 is missing in either h or R. To find the last missing relation can easily take more time than *the entire previous run* [emphasis added]"

The results for our implementation were consistent with this. This differs from the experience of Jacobson who found that for his implementation $\widetilde{h}$ was often the true value of h. Discussing imaginary quadratic fields in [Jac99b, Section 5.2.2] he lists the values of $\widetilde{h}/h$ and notes:

> "In only nine out of the ninety-five examples listed in these tables was the index greater than 1."

In all these examples the multiple of h was 2.

To find additional relations to obtain the true class number and regulator our implementation does the following:

- If we have $\widetilde{h}\widetilde{R} \geq 2\widetilde{hR}$ we check the number of reserved random relations and load 20 of them.

- Otherwise, we set $E = 2E$ and restart the algorithm from the beginning.

In some cases it is possible to deal with the problem of $\widetilde{h}$ and $\widetilde{R}$ being small multiples of their true values without generating additional relations. Biasse-Fieker [BF12] propose two methods they call saturating the unit group and saturating the class group. In the course of writing our implementation, we implemented these approaches and experimented with them for quartic fields with $|\Delta|$ of 100 to 200 bits. In these experiments saturating the unit group was fairly successful in extending the lattice when R was a very small multiple of the true value, no more than 10, say. In the case when the multiple is 2 it was frequently successful. As the multiple is frequently no more than 10 this was promising. On the other hand, saturating the class group was rarely successful. Separately, a challenge with both approaches is that there is not a cost-effective way to know in advance if they will succeed. Nor is it possible to predict from the number of relations in M how large $\widetilde{h}/h$ and $\widetilde{R}/R$ will be. Of course this is not to suggest the saturation techniques are not beneficial, but it is evidence of the difficulty of implementing them in a way that is efficient. Due to this difficulty, and the additional complexity that would be introduced into our algorithm by using them, our implementation does not use either approach.

5.2.5 Pseudocode

Algorithm 3 gives pseudocode for the algorithm implementation we have described in the previous sections.

Algorithm 3 QuarticClassGroup

```
Input:
    $f$ : defining polynomial that is $\lambda$-reduced for $\mathbf{K}$ with $\lambda \leq \frac{1}{6}$
    $|\Delta|$ : absolute value of $\mathbf{K}$'s discriminant
    $E$ : sieve radius
    $t$ : the number of primes to sieve
Output:
    $h$ : the class number
    $R$ : the regulator
1: Compute $\mathbf{Z}$-basis of $\mathcal{O}_{\mathbf{K}}$, $\mathrm{FB}_{\mathbf{K}}$, $\widetilde{hR}$, $k_{\mathrm{req}}$ as described in Section 5.2.1
2: Generate relations using Algorithm 2 as described in Section 5.2.2
3: Load and process the relations as described in Section 5.2.3
4: if $\widetilde{h} = 0$ or $\widetilde{h}\widetilde{R}$ does not satisfy $\widetilde{hR} < \widetilde{h}\widetilde{R} < 2 \cdot \widetilde{hR}$ then
5:     Obtain more random relations as described in Section 5.2.3
6:     Recompute $\widetilde{h}$ and $\widetilde{R}$ by the methods in Section 5.2.4
7: end if
8: return $h$, $R$
```

5.2.6 Implementation Correctness

The testing we did to validate this implementation gives correct results has two parts. First, using `Magma` we computed the class group structure and regulator of a number of fields with discriminants between 100 and 200 bits. Further details on how these fields were constructed can be found in Section 5.3.2. Then after running our self-initialization implementation on a number field we checked our class group structure and regulator against the values from `Magma`.

Second, we added assert statements to validate the program's data in the course of execution. For example, after each important step in the linear algebra stage we expect that $\mathrm{FB}_{\mathbf{K}}^{\mathbf{v}}$ is a principal ideal generated by γ for every relation $(\mathbf{v}, \gamma)$. We use assert statements to verify this for every relation. These assert statements are enabled for a testing build of the implementation which we run regularly. Of course, the production build, the build we use when collecting data on the implementation's performance, disables these assert statements so there is no performance overhead. The benefit of using assert statements rather than unit tests for ensuring correctness is that assert statements are better adapted to code that is changed frequently. Assert statements work on "live" data and so when the code changes only the assert statement itself must be changed. In contrast, unit tests feed test data to a function in order to put the program into the particular state needed for the test. Thus changes to the code very often require changes to the test data. The time spent on updating unit tests can become very large when there are many tests.

5.3 Parameter Tuning

5.3.1 Methodology

Our goal is to take a positive integer k and return f that defines a quartic field $\mathbf{K}$ so that $|\Delta| \approx 2^k$. As the purpose of these polynomials is for comparing the performance of our implementation for arbitrary number fields we try to ensure that the polynomials are not biased in favor of a particular approach to relation generation.

One way to construct such polynomials is to generate integral polynomials $f = \sum_{i=0}^{4} c_i X^i$ so that the ith coefficient c_i is chosen from $(-A, A)$ for some bound A. To obtain $|\Delta| \approx 2^k$ we can take $A = 2^{k/(2\cdot(d-1))} = 2^{\frac{k}{6}}$ with $d = 4$. In most cases this polynomial is irreducible over the rationals and therefore defines a quartic number field. It can happen that f does not define a degree 4 number field, however, this becomes unlikely as A grows larger. For A larger than say 2^{20} it happens rarely.

This construction is used because it allows us to construct defining polynomials that generate a number field of the discriminant size we want. For many applications we will not have a polynomial of bounded coefficients but will instead have polynomials that have been reduced by `polredabs` or a similar reduction algorithm. For example, the canonical defining polynomials used by the `LMFDB` are computed using `polredabs`. To obtain polynomials of this form we apply `polredabs` to f. Like the `polredabs` polynomials we tested from the `LMFDB`, these polynomials are all λ-reduced with $\lambda \leq \frac{1}{6}$.

5.3.2 Parameter Selection

In this section we explain how parameters such as B were selected. Parameter selection was done so that the user does not need to set these parameters manually.

Selection of tuning parameters for any algorithm can be viewed as a search over a multidimensional space. We have observed two challenges to tuning self-initialization implementation: First, there is a high degree of interdependence between the parameters. Changing the value of one parameter means that all the other parameters must be updated. Second, the variance of the optimal parameter choices between number fields is large. As an arbitrary example of the variance in the optimal size of a single parameter we can consider the size of interval used for the special-q approach, for two of our test number fields of the same size $|\Delta|$, we have observed cases for which the best choice for the size of the interval of special-q, with all other parameters held constant, differs by a factor of 100. Like Jacobson [Jac99b] our parameter selection is informed by our experience running the algorithm thousands of times as it was created.

This benchmarking was done on quartic fields of discriminant size between 81 and 193 bits. Runs that were expected to take more than five minutes were monitored manually by examining the implementation's logs during execution so that the total running time relative to the best previous run could be estimated (in an ad-hoc manner). If this predicted performance seemed much worse than the previous best run then the run was considered unsuccessful and was discarded. Successful runs had a total running time ranging from seconds to more than one day.

Once development of the implementation was completed, final selection for each parameter was made based on simple criteria described below.

A great deal of research time was invested into refining our implementation with the different optimizations we have discussed above. This limited the time available for tuning parameters. Another difficulty is that the server our implementation was tested on is shared by many users so we cannot overutilize the system by doing more than a few runs at a time.

These parameters are the best we could find in the available time. Like the tuning parameters for any algorithm, the total running times that are found when using them is dependent on: the hardware that we tested on, the compiler version used to compile the software libraries we used, and the versions of the software libraries that were used.

As the optimal parameters vary greatly even for fields with the same size $|\Delta|$ the default parameters can be customized by the user. When optimizing the default parameters we narrowed down the space of all possible inputs to focus on what we believe are the most important cases. The default parameters are tuned for discriminants of 100 bits or more, although the implementation is applicable to smaller discriminants with reduced efficiency. We chose this focus because, as discriminant size increases beyond this threshold, our self-initialization approach becomes more effective. We recall the following facts from Section 3.4.2 and Section 3.6 respectively: First, in quadratic fields `Magma` switches from enumeration-based relation generation to their SIQS implementation at a 67-bit (20 digit) discriminant size. Second, Biasse-Fieker in [BF12] give data showing that their sieving approach becomes less effective as the degree of the number field is increased. These facts support our hypothesis that for quartic fields, a self-initialization approach in degree four could not be beneficial for discriminants under 100 bits.

First, we consider the choice of smoothness bound B. The relationship between the total time of the algorithm and the size of B was discussed in Section 3.1.1. Recall that B is one of the most important parameters to optimize because it has a large impact on the time for relation generation and the time for linear algebra.

Recall from Section 2.6 that Belabas-Diaz-Friedman show in [BDyDF08] if T is an integer satisfying the inequality (2.7) then the class group of $\mathbf{K}$ is generated by prime ideals of $\mathbf{K}$ having norm strictly smaller than T, under GRH. We let $T_{\min}$ be the smallest T satisfying this inequality. Authors such as Jacobson [Jac99b] and Cohen [Coh93] express the choice of B as a function of the best available bound for a minimal B under GRH which for them was the Bach bound and for us is $T_{\min}$. We can compute $T_{\min}$ in `Magma` using the function `GRHBound`. Although it is theoretically possible that $T_{\min}$ can be larger than the Bach bound $12(\log(|\Delta|))^2$, we found $T_{\min}$ was much smaller than $12(\log(|\Delta|))^2$ for all the number fields we tested. Our observations are in line with a rule of thumb mentioned by Belabas-Diaz-Friedman that $T_{\min}$ is often more than 20 times smaller than the Bach bound (based on sizes of $|\Delta|$ where index-calculus is the algorithm of choice for computing class groups). Thus $T_{\min}$ is the state-of-the-art choice for the minimum B we can choose in order to show that the entire class group is generated.

Let c_B be the constant so that $B = c_B \cdot T_{\min}$. The number fields we tested are given in Table 5.1. We selected three number fields with $k = 16 \cdot i$ for i from 6 to 10 in the construction in Section 5.3 The first column of number fields has unit rank $r = 1$, the second column of number fields has unit rank $r = 2$, the third column of number fields has unit rank $r = 3$. We varied r as a precaution designed to ensure that the self-initialization implementation performs well for all three possible unit ranks. We later learned that r has little correlation with the total running time. Tuning B without the rank restriction should therefore give similar results to what we suggest here. The field labels are d followed by the size of the discriminant in bits.

We ran our implementation for three values of c_B for each of the number fields in a row of Table 5.1. The final c_B was selected as the value was optimal for the majority of the three number fields in the row. To select c_B we started with the number fields of smallest discriminant, those in the first row, we ran the algorithm for c_B in $[0.5, 1, 1.5]$ which we selected based on computational experience with the implementation.

$r = 1$ Fields	$r = 1$ $T_{\min}$	$r = 2$ Fields	$r = 2$ $T_{\min}$	$r = 3$ Fields	$r = 3$ $T_{\min}$	i	Chosen c_B
`d98`	2771	`d96`	2654	`d101`	1877	6	1
`d114`	6221	`d113`	4929	`d106`	3234	7	1
`d134`	9438	`d128`	7856	`d126`	7578	8	1
`d144`	10917	`d140`	6587	`d142`	9621	9	1
`d161`	13656	`d159`	12213	`d151`	9847	10	1.5

Table 5.1: Selection of Smooth Bound

We found that for $|\Delta|$ of size close to 100 bits $c_B = 1$ but as $|\Delta|$ increased towards 150 bits we found that the optimal c_B switched from 1 to 1.5. We therefore switched to choosing c_B from $[1, 1.5, 2]$ for the fifth row of number fields. For these number fields the best c_B was 1.5.

For example, the smallest total running time for `d161` was found with $c_B = 1.5$ taking just over 11 minutes. The $c_B = 1.5$ time was about 18.5 times faster than the time for $c_B = 1$ (which took about 3.4 hours) and about 4.6 times faster than the time for $c_B = 2$ (which took about 51 minutes).

We turn to the selection of the number of rational sieving primes t. These are the primes chosen from $\{p \in \mathrm{FB}_{\mathbf{Z}} : p \nmid [\mathcal{O}_{\mathbf{K}} : \mathbf{Z}[\theta]], \quad \mathfrak{p}^{(0)}\mathfrak{p}^{(1)} \mid (p), \quad \mathfrak{p}^{(0)} \neq \mathfrak{p}^{(1)}\}$ which we then use in the construction of the 2^t initial ideals for sieving. We let

$$A_s = \prod_{i=1}^{s} p_i$$

where the product is over the rational sieving primes, ordered by their size. The smallest integer s' satisfying the inequality

$$A_{s'} > \frac{1}{6}||f||$$

is the largest feasible choice of t that allows us to choose t rational sieving primes with their product close to $\frac{1}{6}||f||$. As we are working with a general case defining polynomial, the right hand side of this inequality is approximately $\frac{2^4}{6}|\Delta|^{\frac{1}{6}}$. In our computations we have found that a larger t is beneficial for the efficiency of our self-initialization implementation. Unfortunately for our self-initialization approach, the growth of s' as a function of $|\Delta|$ is somewhat slow. For this reason, we always set t

equal to s'.

We now turn to some parameters that are less important for the total running time. For the special-q approach we let c_q be the size of the interval of special-q. For selecting c_q we considered $[25, 50, 75]$ which we selected based on computational experience with the implementation. Let c_q be the size of the interval of special-q primes used. We selected a value by running the implementation on `d161` for these different q values. The best total running time was for $c_q = 50$. This was 7.1% faster than $c_q = 25$ and 5.1% faster than $c_q = 75$.

For LB, the smoothness bound for the large prime, we tried $\log_2 LB = \lfloor 0.1 \cdot \log_2(|\Delta|) \rfloor + c$ for c in $[3, 5, 7]$ which we selected based on computational experience with the implementation. We selected a value by running the implementation on `d1611` for these different q values. The best total running time was for $c = 5$. This was 20.6% faster than $c = 3$ and 35.0% faster than $c = 7$.

Chapter 6

Computations

Contents

To show the efficiency of our self-initialization approach, we give computational results on the self-initialization implementation (Algorithm 3) and also make comparisons of our implementation with `Magma`'s implementation (Algorithm 4) as it is the current state-of-the-art. In Section 4.2 we categorized λ into two cases, special and general. The general case is defined as λ approximately equal to $\frac{1}{6}$, while the special case corresponds to λ signficantly less than $\frac{1}{6}$. The computational results in this chapter are accordingly divided into these two scenarios.

The general case is addressed in Section 6.1.2 and Section 6.1.3. Our main priority is to show that our approach to relation generation is effective. In Section 6.1.2 we consider relation generation using the metric of *time per relation* as a way to measure the quality of a relation generation approach. This is defined in terms of the set of "useful" relations, the ones included in the relations matrix M to make it full rank. Its significance is that it quantifies the speed that useful relations are generated.

Jacobson [Jac99b] found the time per relation for his self-initialization approach was significantly smaller than the time per relation for all prior approaches. Thus, our main hypothesis was that our self-initialization method would have a smaller time per relation than prior work. The validation of this hypothesis is an important step toward the potential future integration of self-initialization into index-calculus algorithms. Our self-initialization relation generation has better time per relation than `Magma` and the difference grows larger as $|\Delta|$ increases. This finding suggests that, with further optimization efforts, our approach could outperform `Magma` in terms of efficiency.

The quadratic sieving literature, for example [Sil87], gives a strong indication that the strategy used for sieving with multiple polynomials significantly impacts performance. Thus, it would be interesting to know how the analogue of MPQS described in Section 4.3 compares to other approaches in the literature using multiple sieving polynomials. We believe our MPQS approach will produce ideals of smaller norm than the approaches in the literature that require finding elements of small height described in Section 3.5.2. Unfortunately, there are no implementations of these approaches that are available to benchmark against.

Another aspect of relation generation time is the time to initialize the sieving data. By comparing the time spent on initializing the sieving for our self-initialization approach to the time spent on initializing the sieving for our MPQS analogue we show that self-initialization is a beneficial optimization for our approach.

In Section 6.1.3 we give results on the total running time of our approach. The performance of our self-initialization implementation is comparable to `Magma`'s highly-optimized implementation.

For the special case, addressed in Section 6.2, we employ two families of defining polynomials to demonstrate the impact of λ. Defining polynomials of pure number fields are used for this purpose because they allow us to precisely control the value of λ. Comparing two families of number fields with the same structure allows us to isolate the effect of λ on the algorithm's total runtime. The families we consider have values of λ equal to $\frac{1}{12}$ and $\frac{1}{8}$, respectively. The results we give show that our self-initialization implementation significantly outperforms `Magma`'s when $\lambda = \frac{1}{12}$ for discriminants with more than 150 bits. For $\lambda = \frac{1}{8}$ our approach was predicted to be less efficient because of the larger size of the norms $N(\mathfrak{b})$ that are generated, and this is indeed what we observe. Our performance is still better than `Magma` for sufficiently large discriminants.

The hardware for the computations in this chapter is the same as we used in Section 5.1.

For the purposes of our computations we determined the `Magma` program that is most similar to ours. This program, given in Appendix A, takes a defining polynomial as input and returns the class group and unit group.

In order to time our self-initialization implementation and `Magma`'s implementation we use CPU time. As we do not consider other measurements of time, we refer to "CPU time" simply as "time" in the sequel. Units of time such as seconds always refer to CPU time.

6.1 General Case

We now turn to the case of computing the class group of an arbitrary number field. If we are provided with a defining polynomial of λ larger than $\frac{1}{6}$ we can obtain a $\lambda = \frac{1}{6}$ defining polynomial using `polredabs`. Thus we can represent an arbitrary number field using a λ-reduced defining polynomial.

As discussed in this chapter's introduction, in Section 6.1.2 we consider the relation generation stage. The metric we use to demonstrate the efficiency of our self-initialization relation generation is time per relation which we define in Section 6.1.2. In Section 6.1.3 we give timings of our complete self-initialization-based implementation and compare it to `Magma`'s performance.

6.1.1 Methodology

We use the construction from Section 5.3 for the defining polynomials that are used for the computations in both Section 6.1.2 and Section 6.1.3. Recall that this construction starts by constructing a defining polynomial with discriminant size close a target we set and then applies `polredabs` to this polynomial.

In this section we start timings at discriminants of roughly 80 bits. This is the minimum number of bits for which we think an self-initialization approach could possibly be useful.

We note that these polynomials all happened to have the Galois group of their splitting fields equal to S_4.

6.1.2 Results - Relation Generation

In order to demonstrate the efficiency of our approach to relation generation we use time per relation:

$$\text{Time per relation} = \frac{\text{Time to generate relations so } M \text{ has full rank}}{\text{Number of relations in } M}$$

recalling that M is the integer relations matrix. The time in the numerator is the sum of the time to generate the initial relations using self-initialization (recall that these must generate a matrix whose rank is 97% of $|\,\mathrm{FB}_{\mathbf{K}}\,|$) and the time to generate additional relations to reach full rank. The relations that we add to M are used because not every relation is useful for making progress towards getting a set of relations that spans Λ, for example, duplicate relations do not help at all. A set of relations that give an M of full rank are certainly useful. The reasons we consider time per relation are: first, that the speed of generating useful relations is crucial to the success of a relation generation approach. Second, we need a metric that can be easily computed for `Magma`'s implementation as well as our own and this is possible for time per relation using information in `Magma`'s logs.

Another goal of our investigation of the general case is to determine how time per relation is affected by the height of the sieving polynomials and the unit rank of the number field.

Time Per Relation

As we have mentioned, a similar metric to our time per relation is considered byJacobson [Jac99b] for quadratic fields. He shows SIQS has a smaller time per relation than alternative approaches. The time per relation for real quadratic fields is given in [Jac99b, Tables 5.5-5.6] and for imaginary quadratic fields in [Jac99b, Tables 5.21-5.22].

Our main hypothesis was that our self-initialization method would have a smaller time per relation than prior work. The validation of this hypothesis is an important step toward the potential future integration of self-initialization into index-calculus algorithms. We give timings of the time per relation of our complete self-initialization implementation and compare it to `Magma`'s time per relation. Our results show that our time per relation is often more than 4 times smaller than `Magma`'s. Furthermore, the ratio of `Magma`'s time per relation over our time per relation increases as $|\Delta|$ increases. Both these findings validate our hypothesis. By this measure, our self-initialization relation generation surpasses that of `Magma`. This finding strongly suggests that, with further optimization efforts, our approach could outperform `Magma` in terms of efficiency.

We felt a comparison of time per relation would be more useful if the same smoothness bound was used for both algorithms. Since we cannot change the B used by `Magma`, we made our approach use the same B as `Magma` rather than the B discussed in Section 5.3. In this section we are trying to make the best case for the use of self-initialization relation generation. For this reason the parameters, such as the sieve radius, were chosen experimentally to minimize the relation generation time.

Both the numerator and the denominator of the time per relation have substantial fluctuations even between number fields of the same size of $|\Delta|$. Consequently, we expect time per relation to fluctuate a great deal. To reduce the variance we averaged timings from several number fields. As we have mentioned, we start our timings at number fields with a discriminant size of 80 bits. For k starting at $k = 5$ to $k = 12$ we constructed 4 defining polynomials with target discriminant size $16 \cdot k$. Timings are averaged for four number fields with discriminant in the interval $(16k - 8, 16k + 8)$. In addition we ensure that for each k there was at least one number field with each of the possible unit ranks: $1, 2$, or 3.

Table 6.1 gives the time per relation for our self-initialization implementation. In the table, and throughout this chapter, SI refers to our self-initialization implementation. We define G as the set of self-initialization sieving polynomials. We use L to mean the tuple of logs of the heights of the sieving polynomials. That is,

$$L = (\log_2 H(g))_{g \in G}.$$

The meaning of the column header names is as follows:

- "Range of $|\Delta|$" means the parameter k defined above.
- "Time per reln (ms)" is the time per relation.
- "Time (s)" is the relation generation time.
- "Min L" and "Mean L" are statistics about L.

- “$|\,\mathrm{FB}_{\mathbf{K}}\,|$” is the number of primes in the factor base.

Range of $\|\Delta\|$	Time per reln (ms)	Time (s)	Min L	Mean L	$\|\,\mathrm{FB}_{\mathbf{K}}\,\|$
80	1.0	0.4	34.9	36.8	266.8
96	1.3	1.0	40.5	46.1	432.3
112	1.5	2.1	47.3	52.4	710.3
128	4.8	7.8	59.4	62.8	1032.0
144	5.3	12.5	63.7	69.6	1490.8
160	18.9	79.3	76.1	79.4	2235.5
176	28.3	148.5	81.9	89.1	3264.7
192	47.6	298.6	92.9	97.1	4590.0

Table 6.1: SI Time Per Relation

Figure 6.1 shows both the SI and `Magma` time per relation. The data is for the same number fields as in Table 6.1. “TPR” is an abbreviation for time per relation.

We observe that as Range of $|\Delta|$ increases “`Magma` Time per relation” tends to increase at a faster rate than the “SI Time per relation” across the full range of $|\Delta|$. Overall the trend of the time per relation is clear: `Magma`’s is growing much faster than ours. Furthermore, the ratio of `Magma`’s time per relation over our time per relation increases as $|\Delta|$ increases. Both these findings validate our hypothesis. According to this metric our self-initialization relation generation is better than `Magma`’s.

Effect of Minimum Height and Mean Height on Time Per Relation

Regarding the height of the sieving polynomials, our self-initialization approach produces a number of quartic sieving polynomials g. We recall from Section 2.7 that the height of a sieving polynomial $g(X,Y)$ is related to the size of $g(x,y)$ in the region $x \in [-E,E]$ and $y \in [-E,E]$. Indeed we have the upper bound $|g(x,y)| \leq (d+1)H(g)E^d$. Recall that Theorem 4.2 bounds $\max g(x,y)_{x,y\in[-E,E]}$ by roughly $|\Delta|^{3\lambda}E^d$. It follows that the size of $H(g)$ should be close to $|\Delta|^{3\lambda}$. Thus, data on $H(g)$ can be used to corroborate Theorem 4.2.

In the case $\lambda = \frac{1}{6}$ our algorithm is supposed to produce sieving polynomials that have height of roughly $\sqrt{|\Delta|}$. In other words, the mean log height of g should be close to $\frac{1}{2}\log_2(|\Delta|)$. The column

Range of $\|\Delta\|$	SI TPR	`Magma` TPR	(`Magma` TPR)/(SI TPR)
80	1.0	0.8	0.8
96	1.3	1.1	0.9
112	1.5	5.3	3.5
128	4.8	15.1	3.1
144	5.3	20.6	3.9
162	18.9	69.4	3.7
178	28.3	127.1	4.5
194	47.6	228.6	4.8

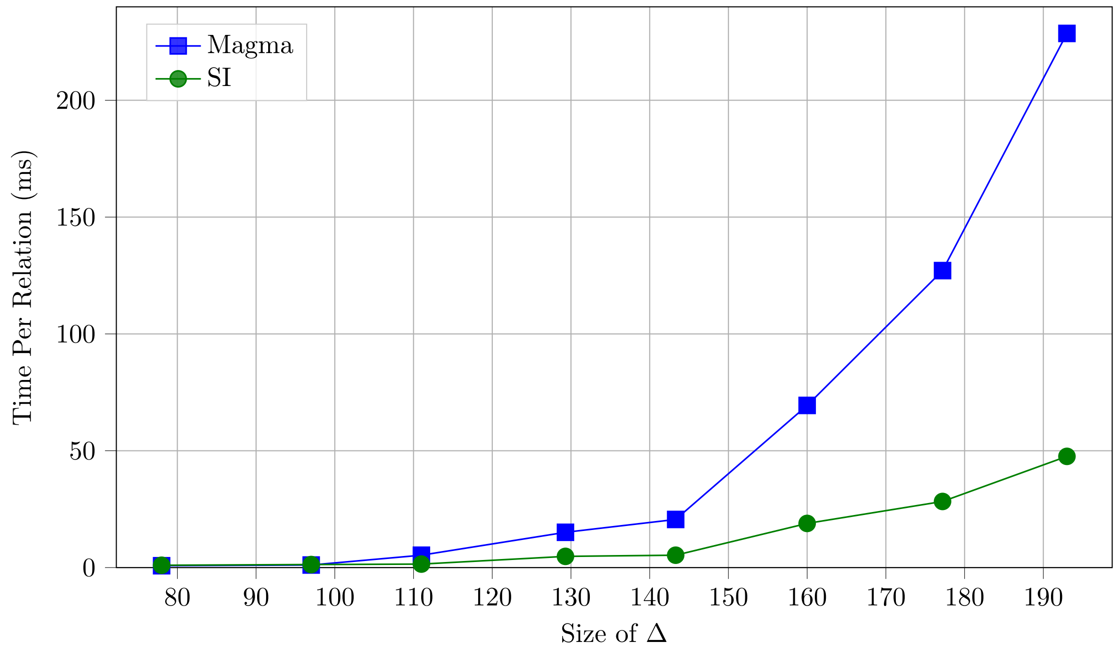

Figure 6.1: Time Per Relation

in Table 6.1 giving the mean log of the height of the sieving polynomials shows that this is indeed the case.

Figure 6.2 is derived from Table 6.1. Recalling that L is the set of logarithmic heights of the sieving polynomials, this figure shows the relationship between “Mean L”, “Min L”, and “Time per reln (ms)”.

Figure 6.2 shows the relationship between these quantities. This figure shows that on average the time per relation increases monotonically with both the mean logarithmic height and the minimum logarithmic height.

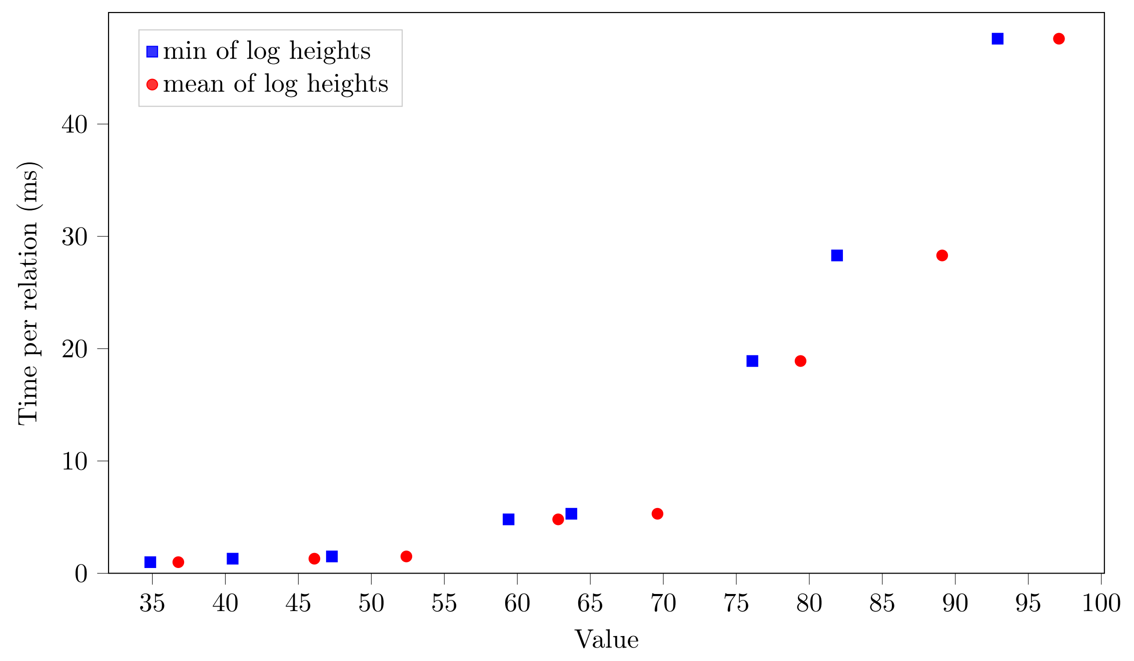

Figure 6.2: Min and Mean Log Height vs. TPR

Effect of Unit Rank on Time Per Relation

In order to assess the effect of the unit rank on the time per relation we split the number fields from Table 6.1 by unit rank. We had no prior belief that the unit group should have a relationship with the time per relation but only consider this because the unit rank is an important invariant of the number field.

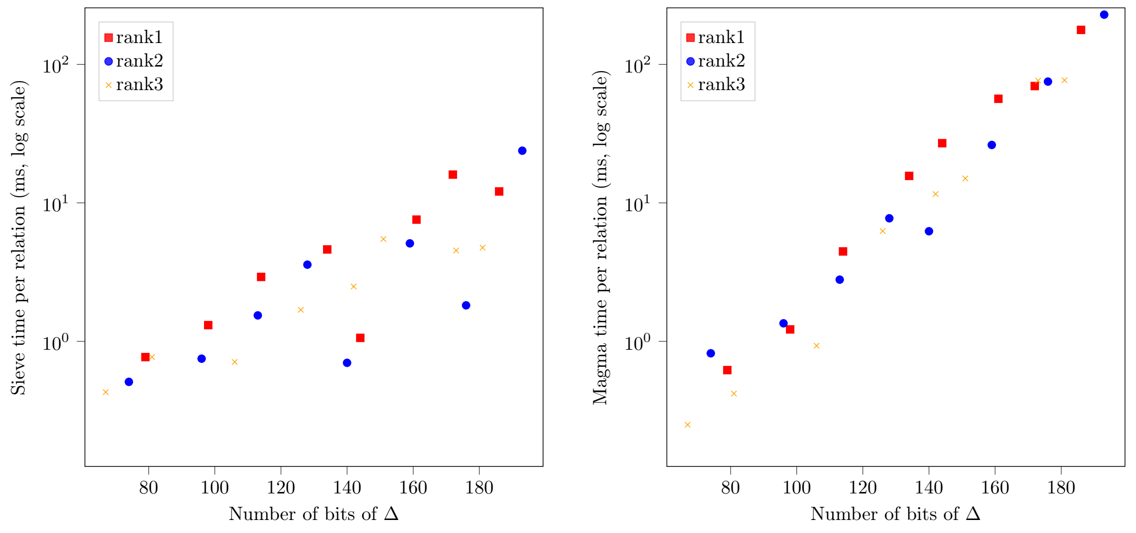

Figure 6.3: Unit Rank vs TPR

Figure 6.3 shows the resulting time per relation for these three possible unit ranks. The graph on the left shows the results for our self-initialization approach. It can be seen that the unit rank of the number field has a negligible impact on the time per relation. This result is in line with what Jacobson found in the quadratic case in [Jac99b]. In [Jac99b] Table 5.4 and Table 5.19 have a column for the number of elements tested for smoothness. The unit rank has a small impact on the number of elements tested for smoothness for small discriminants but the difference fades away as the size of the discriminant grows. Tables 5.40 and Table 5.41 in [Jac99b] give the running time for the relation generation stage. For large discriminants the unit rank of the number field has no discernible impact on these times.

In Figure 6.3 the graph on the right shows the time per relation for `Magma` as a function of the unit rank. This shows the unit rank also has no impact on `Magma`'s time per relation.

Time to Initialize the Sieving Data

In the quadratic case, the use of multiple polynomials within relation generation is an important speedup over sieving a single polynomial. Additionally, self-initialization further enhances this performance. Our self-initialization implementation behaves similarly.

We now consider the additional improvement obtained in our self-initialization approach over the multiple polynomial approach. Unlike [Con97] and [Jac99b], where the motivation for using self-initialization is the high cost of initializing the sieving data, in our implementation the cost of initializing sieving polynomials is a small fraction of the time spent on relation generation. The vast majority of the time is spent on sieving. Thus, we did not expect the performance improvement of self-initialization to be a large percentage of the total time taken by the algorithm.

To do this we draw the same ideals from S_N as are used for self-initialization in Algorithm 2. We use a more direct, but presumably less efficient, method for computing the 2^t sieving elements $[A, \theta - P_i]$ and roots of the sieving polynomials modulo the primes in the rational factor base. In particular, every P_i is calculated using equation (4.10) rather than by using equations (4.17)–(4.18) and roots of the sieving polynomial are calculated with equation (4.13) rather than with equations (4.19)–(4.20). We use the abbreviation MP for this approach and as usual SI denotes the self-initialization approach.

The two approaches only differ in the time to initialize the data for sieving, all other steps of the algorithm run in the same time. For this reason we compare the times to initialize the data for sieving which are given in Table 6.2. The timings are for the same number fields as were used in Section 6.1.2.

Range of $\|\Delta\|$	t	B	SI Init Time (s)	MP Init Time (s)	(MP Time)/(SI Time)
80	2	1246	0.19	0.48	2.5
96	3	2434	0.36	1.07	3.0
112	3	4999	0.66	2.16	3.3
128	4	8291	1.43	5.15	3.6
144	4	11475	1.92	9.82	5.1
160	5	20798	3.24	23.73	7.3
176	5	25525	4.05	36.24	8.9
192	6	31708	5.62	55.28	9.8

Table 6.2: SI vs MP Sieving Data Init Time

The results in Table 6.2 show that self-initialization is beneficial. The performance of both approaches is impacted by B, which determines the number of primes in the rational factor base, and the parameter t.

6.1.3 Results - Total Time

We now turn to considering the running time of the entire algorithm. Table 6.3 gives the parameters used in timing the entire algorithm. These parameters were found by tuning the parameters which we have discussed in Section 5.3.

Range of $\|\Delta\|$	$B/T_{\min}$	$T_{\min}$	$\log_2 LB$	t	c_q	E
80	1	1246.3	13	2	50	512.0
96	1	2434.0	14	3	50	853.3
112	1	4998.7	16	3	50	1194.7
128	1	8718.0	17	4	50	7168.0
144	1	11474.7	19	4	50	3072.0
160	1.5	13470.3	21	5	50	10922.7
176	1.5	16469.0	22	5	50	32768.0
192	1.5	19062.3	24	6	50	43690.7

Table 6.3: SI Parameters

Table 6.4 gives the time for the three most time-consuming stages of the algorithm: self-initialization relation generation (RG), additional relation generation (AR), and linear algebra (LA). For k starting at $k = 5$ to $k = 12$ we used three of defining polynomials with target discriminant size $16 \cdot k$ mentioned in Section 6.1. Timings are averaged for number fields with discriminant in the interval $(16k - 8, 16k + 8)$.

Range of $\|\Delta\|$	RG time (s)	AR Time (s)	LA Time (s)	Total Time
80	1.3	0.7	3.4	5.4
96	2.2	7.4	6.5	16.0
112	9.1	11.2	17.9	38.2
128	69.2	22.4	35.8	127.4
144	100.6	26.4	44.4	171.4
160	602.2	56.8	118.1	777.0
176	3133.0	320.3	230.0	3683.4
192	10968.8	1482.3	415.9	12867.1

Table 6.4: SI Total Time

In Table 6.4 the time for the relation generation and additional relation generation steps is large compared to the time for linear algebra. Comparing the results in this table to Jacobson's results in the quadratic case, a higher percentage of time is spent on relation generation than either the imaginary or real quadratic cases (see Table 5.13 for the imaginary case and Table 5.35-5.36 for the real case). Note that although the times in the table are given in seconds, their equivalent in hours for the larger $|\Delta|$ are shown in Figure 6.5.

We now give timings for `Magma`. Table 6.5 lists parameters used by `Magma`'s implementation as well as the total running time of the time-consuming stages: relation generation and linear algebra.

Range of $\|\Delta\|$	B	$\log_2 LB$	RG time (s)	LA time (s)	Total Time (s)
80	1246.3	13.5	0.2	0.6	0.8
96	2475.0	16.6	0.5	1.1	1.6
112	4998.7	18.3	2.2	2.0	4.3
128	8406.7	19.0	11.0	4.7	15.7
144	11474.7	19.5	21.7	7.3	29.0
160	18131.7	20.1	71.9	23.2	95.2
176	26925.7	21.7	224.5	84.2	308.6
192	37558.7	22.2	700.5	473.6	1174.1
208	56426.0	22.8	3170.4	2398.2	5568.6
224	79825.3	23.3	11790.3	9018.3	20808.6
240	112490.0	23.8	25905.2	28493.8	54399.1

Table 6.5: `Magma` Total Time and Parameters

Figure 6.4 compares the total time of `Magma`'s implementation and the self-initialization implementation. The table lists the total times for `Magma`'s implementation and the total time for self-initialization implementation. We show the total running times that took under 4 hours.

Our initial hypothesis was that our approach would be capable of generating relations that will give us the class group within an amount of time not greatly exceeding `Magma`'s highly-optimized implementation. The timings in Table 6.5 and Figure 6.5 validate this hypothesis.

As `Magma`'s implementation, which we discussed in Section 3.4.2, includes optimizations our self-initialization approach does not have and `Magma`'s implementation has been refined over many years, it is unsurprising that it is faster than our self-initialization approach. With further work on the rest of our implementation, including adding all the optimizations included in `Magma`, we have reason to expect that our new relation generation with superior time per relation will lead to faster overall performance as well.

Range of $\|\Delta\|$	SI Total Time	`Magma` Total Time	`Magma`/SI
80	5.4	0.8	0.15
96	16.0	1.6	0.10
112	38.2	4.3	0.11
128	127.4	15.7	0.12
144	171.4	29.0	0.17
160	777.0	95.2	0.12
176	3683.4	308.6	0.08
192	12867.1	1174.1	0.09

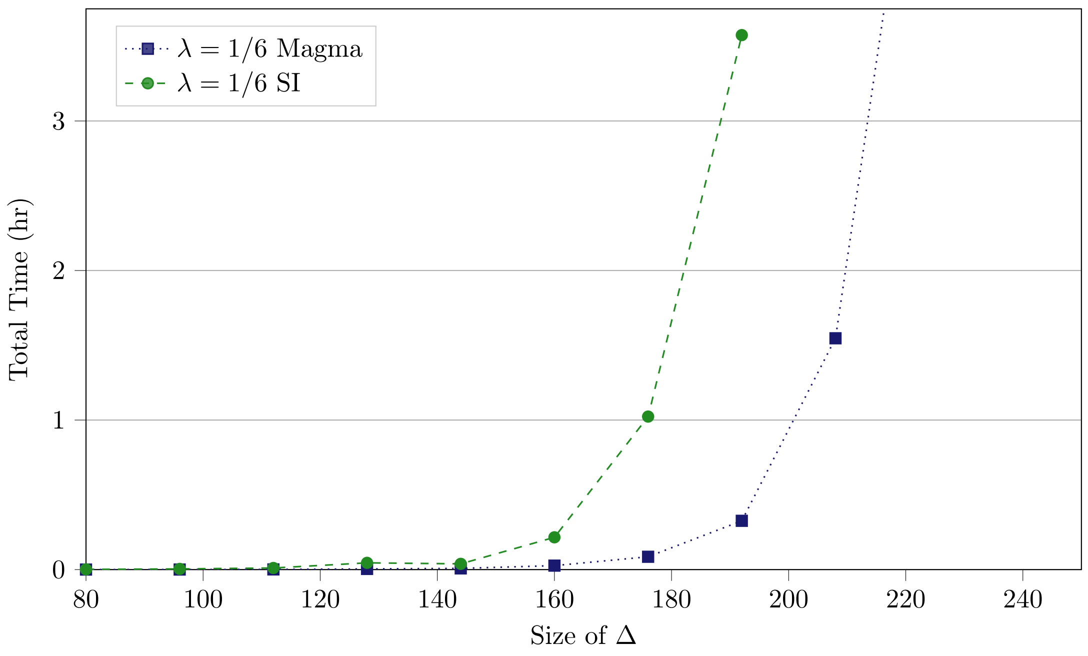

Figure 6.4: SI Parameters

6.2 Special Case

In this section we use pure fields to illustrate how the efficiency of the self-initialization implementation depends on λ. Recall that a field is pure if it can be represented by a defining polynomial of the form $f = X^4 - D$ for an integer D. A subset of these number fields $\mathbf{K}$ are those defined by $f = X^4 - ab^2$ where a is squarefree. Recall from Section 4.1 that we can choose a, b so that λ has any value we like greater than or equal to $\lambda = \frac{1}{12}$.

6.2.1 Methodology

We use the defining polynomial $f = X^4 - ab^2$ where $b = a^\ell$ for two different values of ℓ. First, in Section 6.2.2 we consider the family of defining polynomials with $\ell = 0$ for number fields of different $|\Delta|$ sizes. Our results show that we can do significantly better than the algorithm used by `Magma` in this case.

Given a real number ℓ, a target number of bits k, and a real number λ between $\frac{1}{12}$ and $\frac{1}{6}$ we want to choose a, b so that $|\Delta|$ has roughly k bits and $||f||$ is roughly $|\Delta|^\lambda$. In Section 4.1 we discussed the discriminant of $\mathbf{K}$ (equation (4.3)) and also determined that

$$||f|| \leq 2^5 |\Delta|^{\frac{1+2\ell}{4(3+2\ell)}}.$$

Our approach is to choose a random $a \in \mathbf{Z}_{>0}$ that is close to $2^{\frac{k}{(1+2\ell)}}$ and a random $b \in \mathbf{Z}_{>0}$ to be close to a^ℓ. We test the polynomial for irreducibility and stop when the first irreducible polynomial is found. In practice, we get an irreducible polynomial after only a few attempts.

As an aside we mention a property of `polredabs` when applied to polynomials of this form. We found `polredabs`$(X^4 - ab^2) = X^4 - ab^2$ for all defining polynomials used for computations in this section. This behavior is certainly not true of reduced polynomials in general, it is specific to these defining polynomials and to small values of λ. We note that $\lambda = \frac{1}{8}$ is the largest λ considered in this section.

As discussed in Section 5.3 we chose to tune the parameters of the self-initialization implementation for the general case. However, we have observed that the best B depends on λ as well as $|\Delta|$ and so timings for the special case using this B would not be an accurate representation of what is possible with our self-initialization approach. For this reason we have used a somewhat smaller B for computations in the special case than the B selected for the general case in Section 5.3.

Recall that the `Magma` algorithm we compare against is given in Appendix A. Some information can be obtained from the logs including the smoothness bound and the total time of `Magma`'s implementation which we reference in this section.

6.2.2 Results for $\lambda = \frac{1}{12}$

The first family we consider is when $\ell = 0$ and so the defining polynomial is $f = X^4 - a$ where a is squarefree. In this case we choose a higher range of $|\Delta|$ than we chose for the general case because there is a crossover point where our algorithm becomes more efficient than `Magma`'s. We aim to begin the range slightly below this crossover point. The data in this section are presented as averages of three number fields.

Table 6.6 gives the value of B used by each of the three algorithms.

- "$T_{\min}$" is the Belabas-Diaz-Friedman smoothness bound discussed in Section 2.6.
- "SI $B/T_{\min}$" gives the multiple of $T_{\min}$ that we used in the timing results discussed later in this section.
- "`Magma` $B/T_{\min}$" gives the multiple of $T_{\min}$ used by `Magma`.

For the self-initialization implementation we found it effective to set $B = T_{\min}$ for all cases we tested for $\ell = 0$.

Range of $\|\Delta\|$	`Magma` $B/T_{\min}$	SI $B/T_{\min}$	$T_{\min}$
144	1.8	1	7477
176	2.1	1	13685
208	2.6	1	29831
240	3.4	1	43496

Table 6.6: $\lambda = \frac{1}{12}$ Smoothness Bound Data

Table 6.7 shows the connection between $||f||$, the heights of the self-initialization sieving polynomials, and the total time. We let G be the set of self-initialization sieving polynomials. We use L to mean the tuple of the log of the height over the set of sieving polynomials. That is,

$$L = (\log_2(H(g)))_{g \in G}.$$

For this family of defining polynomials we have $|\Delta|^{3\lambda} \approx |\Delta|^{\frac{1}{4}}$. The column "(Mean L)/($\log_2 |\Delta|$)" gives us the value z so that $H(g) \approx |\Delta|^z$ for a typical sieving polynomial g. From our previous discussion of $H(g)$ we expect $z \approx \frac{1}{4}$. The table confirms this, suggest the value will eventually converge to $\frac{1}{4}$.

We expect that Mean L is correlated with the time for the relation generation phase and hence of the entire algorithm. The table demonstrates this correlation.

Range of $\|\Delta\|$	$\log_2(\|\Delta\|)$	$3\log_2 \|\|f\|\|$	Mean L	(Mean L)/($\log_2 \|\Delta\|$)	Time (s)
144	146.5	40.8	41.7	0.29	34.4
176	187.6	49.2	49.8	0.28	60.1
208	218.0	57.0	57.6	0.27	396.5
240	249.5	64.9	65.7	0.26	1415.5

Table 6.7: $\lambda = \frac{1}{12}$ Height and Time Statistics

Table 6.8 gives the time per relation for SI and for `Magma`. The times are consistently better for self-initialization than for `Magma`. Comparing this with general case we see the time per relation is smaller for the $\lambda = \frac{1}{12}$ case.

Range of $\|\Delta\|$	SI TPR	`Magma` TPR	(`Magma` TPR)/(SI TPR)
144	0.3	3.1	15.4
176	0.6	10.1	15.5
208	0.8	82.4	87.8
240	1.2	154.9	134.4

Table 6.8: $\lambda = \frac{1}{12}$ Time Per Relation

Table 6.5 gives the time for the whole algorithm for pure number fields for self-initialization and `Magma`. Due to the choice of discriminant range, `Magma` is faster for the first row of averaged discriminants but self-initialization is more efficient for the remaining rows. The final column gives the ratio of the `Magma` time over the self-initialization time (larger is better for us). The fact that this ratio grows to over 40 demonstrates the value of our self-initialization approach.

6.2.3 Results for $\lambda = \frac{1}{8}$

We now turn to our second family. By taking a larger value of ℓ we get a family of reduced defining polynomials with a larger λ. The family with $\ell = 0$ illustrates the case where $\lambda = \frac{1}{12}$ is as small as we know how to make it. We choose this second family to illustrate the intermediate behavior. For the second case the value of ℓ was chosen so that λ is the mean of these two extremes. The mean of $\frac{1}{6}$ and $\frac{1}{12}$ is $\frac{1}{8}$. We see that taking $\ell = \frac{1}{2}$ gives a defining polynomial with this λ. This gives a second family of pure fields to compare the first against. As both fields are pure, this comparison isolates the effect of different λ on total running time.

Data in this section is an average of two number fields. The range of $|\Delta|$ used is smaller because

Range of $\|\Delta\|$	`Magma` Time (s)	SI Time (s)	`Magma`/SI
144	26.0	34.4	0.8
176	280.0	60.1	4.6
208	7393.8	396.5	18.6
240	61362.2	1415.5	43.4

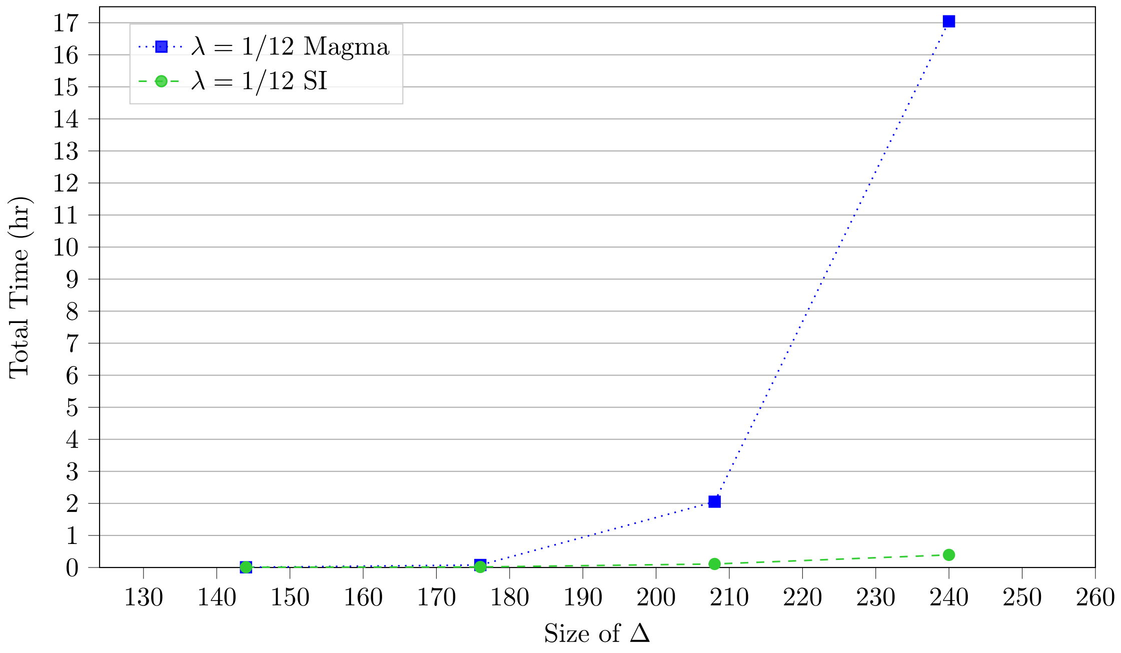

Figure 6.5: $\lambda = \frac{1}{12}$ `Magma` vs. SI

of the larger computation time vs. $\lambda = \frac{1}{12}$.

Table 6.9 gives the value of B in terms of $T_{\min}$. Our $T_{\min}$ is the value computed by `Magma` but starting around 100 bits `Magma` starts using B larger than $T_{\min}$. As was the case for $\ell = 0$, the size of the B used by `Magma` grows with $|\Delta|$. Again the range of $|\Delta|$ was chosen so that `Magma` is slightly faster for the first few discriminants.

We found it was effective to set $B = T_{\min}$ for the number fields with $|\Delta|$ with size less than 200 bits. Above 200 bits we used $B = 1.25 \cdot T_{\min}$. Comparing with Table 6.6 we see that the size of $T_{\min}$ is roughly the same. For Table 6.6 the most effective B was always $B = T_{\min}$.

Range of $\|\Delta\|$	`Magma` $B/T_{\min}$	SI $B/T_{\min}$	$T_{\min}$
144	1.5	1	8545.5
176	1.8	1	28304.0
208	2.3	1.25	36426.5

Table 6.9: $\lambda = \frac{1}{8}$ Smoothness Bound Data

Table 6.10 shows the connection between $||f||$, the heights of the self-initialization sieving polynomials, and the total time. In this case we have $||f||^3 \approx |\Delta|^{3\lambda} = |\Delta|^{\frac{3}{8}}$ and so we expect z to be approximately $3 \cdot \frac{1}{8} = 0.375$. We see this is indeed the case.

Range of $\|\Delta\|$	$\log_2(\|\Delta\|)$	$3\log_2 \|\|f\|\|$	Mean L	(Mean L)/($\log_2 \|\Delta\|$)	Time (s)
144	143	55.5	56.2	0.39	53.9
176	198	75.3	75.8	0.39	890.8
208	229	87.1	86.9	0.39	11546.1

Table 6.10: $\lambda = \frac{1}{8}$ Height and Time Statistics

Table 6.11 compares the time for the whole algorithm for self-initialization and `Magma`. Each row gives the time for the full algorithm for `Magma` and the self-initialization implementation. As we have mentioned, `Magma` is faster for the first few discriminants but self-initialization becomes more efficient as $|\Delta|$ grows.

As expected, the total running times for the self-initialization implementation are larger for $\lambda = \frac{1}{8}$ than for the $\lambda = \frac{1}{12}$ case. In contrast, the total running times for `Magma` are worse than for the general case. This may simply be due to an unlucky choice of number fields in the sample for $\lambda = \frac{1}{8}$.

The final column gives the ratio of the `Magma` time over the self-initialization time (larger is better for us). The trend of these ratios is upward but as expected the rate of increase is slower than in Table 6.7.

Range of $\|\Delta\|$	`Magma` Time (s)	SI Time (s)	`Magma`/SI
144	22.8	53.9	0.4
176	1282.8	890.8	1.5
208	23343.3	10531.1	2.1

Table 6.11: $\lambda = \frac{1}{8}$ `Magma` vs. SI

6.2.4 Comparison of Both

Finally, Figure 6.6 shows the total running time for $\lambda = \frac{1}{6}$ and for the $\lambda = \frac{1}{12}$ pure fields from Section 6.2.

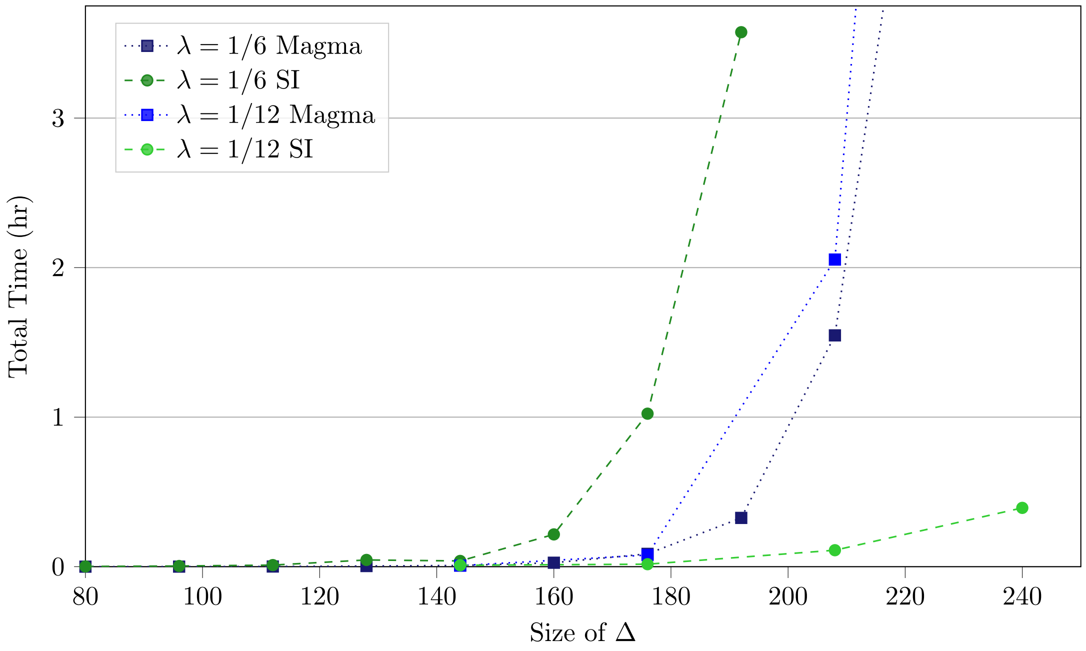

Figure 6.6: Special and General Total Time

Regarding `Magma`'s implementation, the figure shows that the times for `Magma`'s implementation are slightly worse for $\lambda = \frac{1}{12}$ then for $\lambda = \frac{1}{6}$. The difference is not very large, suggesting that `Magma`'s implementation does not depend on λ.

As we have discussed, self-initialization is not faster than `Magma` in the general case but the figure illustrates that we can beat `Magma` by a large amount for special cases.

Chapter 7

Conclusion and Future Work

In this thesis a new method of generating relations using self-initialization was developed. Our new relation generation has better time per relation than the current state-of-the-art, `Magma`. Furthermore, the improvement increases as discriminant size increases. Our implementation of the complete algorithm, including the improved relation generation, is faster than `Magma` for the special case. In the general case our performance is comparable to `Magma`'s. Our results are clear evidence that with further work to incorporate all the optimizations that exist in `Magma` we would achieve an implementation that is faster than `Magma`'s.

There are a large number of possible directions for future work, both in terms of improving the new relation generation method and speeding up the linear algebra and other aspects of the algorithm. Initially, we used `Flint`'s algorithms for linear algebra which made the linear algebra stage the computational bottleneck. Later, we switched to Magma's linear algebra to overcome this issue, a change that improved performance, as shown in the timings in Chapter 6. Currently, the bottleneck has shifted to relation generation.

For the remainder of this section we discuss some potential areas of improvement. We first consider improvements to the relation generation used in Algorithm 1.

One direction is algorithmic improvements to sieving. In [Kle16] a self-initialization approach for quadratic fields is given that handles large factor base primes. This approach identifies relations involving these primes more efficiently than ordinary sieving. This self-initialization approach has recently been generalized to orders of cubic number fields by Luo [Luo24]. According to him, adapting the method to quartic number fields should be possible. Future work could explore the use of this approach.

Another direction is adapting the sieving program `las` from `CADO-NFS` ([Tea23a]) to the problem of computing class groups. `las` has been developed for the computational problems of integer factorization and discrete logarithms modulo a prime thus the relations generated by `las` have two "sides", called the algebraic side and the rational side. We only need to sieve a single side, but we need to accommodate `las` by passing parameters for both sides. An adaptation of `las` could remove the code dealing with the second side.

Our approach could also be extended to allow more large primes to be used. In the random relation generation stage we used one large prime. In the targeted relation generation stage we used zero large primes. In the context of computing class groups of quadratic number fields, two large primes are used by [BJS10] and three large primes are used by [BKV19]. Modern integer factorization algorithms use many large primes [BGG+20] (`las` supports an arbitrary number of large primes). Considering the use of large primes for integer factorization, we suspect the size of the discriminant is the most important factor in determining the optimal number of large primes. It seems likely that two or more large primes would be beneficial for sufficiently large discriminants. Given these observations, integrating more large primes appears to be a promising direction. A somewhat related idea that could be experimented with is Bernstein's batch smoothness testing algorithm [Ber04]. This approach was found to be beneficial by Biasse-Jacobson [Bia10].

Another idea to explore is the use of hybrids of distinct relation generation methods. Biasse-Fieker [BF12] generate an initial set of relations using sieving followed by additional relations with enumeration. They found this approach to be more efficient than relying on sieving alone. Such a hybrid sieving approach could be implemented as they did, by writing code to integrate with the `pari` library in order to leverage its enumeration relation generation.

It may be beneficial to extend the algorithm so that more sieving polynomials are used. Recall that the number of rational sieving primes is t and from these we can construct 2^t initial ideals for sieving. We choose t as the smallest integer so that the product of the first t primes in S_N is larger than $\frac{1}{6}||f||$. We can always choose parameters such as the sieve radius E so that enough relations are obtained from this set of polynomials. However, using more sieving polynomials may make the algorithm more efficient. An approach used by Jacobson that is worth exploring is to repeat self-initialization on sets of different primes. Rather than choosing a single set Q of t prime ideals in S we choose multiple sets Q of t. Each of these sets will contribute 2^t sieving polynomials. Following

Jacobson, we can do this by fixing a subset $Q' \subset S_N$ of t' prime ideals from which the sets Q will be chosen, giving us a total of $\binom{t'}{t}$ different sieving polynomials. To simplify the selection of the subsets Q so that the product of the primes in Q is close to $\frac{1}{6}||f||$ each of the prime ideals is chosen so that $N(\mathfrak{p}) \approx (\frac{1}{6}||f||)^{\frac{1}{t}}$. While this approach does not increase the benefit of self-initialization, which is a function of B and t, it extends the benefit to a larger number of sieving polynomials. The additional parameter t' should therefore improve the algorithm's performance.

A second way the number of sieving polynomials could be increased is by taking advantage of more than two degree one prime ideals above the t rational primes. It is possible that an alternative approach to choosing the set of prime ideals would allow more sieving ideals to be used. An alternative using fully-split primes could be explored. Rather than the "base-two" Gray code our self-initialization approach uses, this variation would use a base-four Gray code to enumerate the sieving data efficiently. This would give us 4^t initial ideals rather than 2^t. A downside of using fully-split primes is that there are usually few fully split primes in a factor base. The rarity of fully split primes can be made somewhat more precise using the Chebatorev density theorem which we discussed in Section 2.4.1. Recall that the tuple of the degrees of the prime ideals is called the decomposition type. This result tells us that the asymptotic density of a prime decomposition type is $|C|/|G|$ where G is the Galois group and C is the conjugacy class of the cycle type corresponding to that prime decomposition type. For a field with Galois group S_4 the density of fully-split primes less than B will be about $|C|/|G| = 1/24$. The set of rational primes that can be used for sieving are the ones with decomposition types $1, 1, 2$ and $1, 1, 1, 1$ (the fully-split primes) so this set has density $6/24 + 1/24 = 7/24$. In summary, our current self-initialization method allows us to use more rational primes for sieving than this alternative fully-split approach but on the other hand it gives fewer initial ideals. Although it is not clear to us which approach would perform better, this variation merits further investigation.

Up to this point, our focus has been on enhancements to relation generation. We will now consider a number of improvements applicable to the overall algorithm.

In the structured Gaussian elimination stage `Magma` leverages efficient sparse integer matrix linear algebra. A full redesign of our integer matrix linear algebra to only use `Magma`'s sparse linear algebra rather than a combination of `Flint` and `Magma` would make the implementation faster.

Another aspect that could be improved is computing valuations of an element in a number field at a prime ideal. At present, we employ the valuation algorithm of `PARI/GP` for this purpose.

Although computing valuations is not a bottleneck for our implementation, a more adapted approach to computing valuations could have some benefit. In particular, leveraging precomputations may improve the efficiency.

Alternatives to `Magma` could be experimented with for the time-consuming linear algebra algorithms. The HNF could be computed with the implementation of [PS13] which is available in the library `hnfproj`. For computing the kernel of the relations matrix there is an implementation of [CS05] which is available as the Integer Matrix Library `IML`. While neither of these libraries has been updated for some time, and their performance relative to `Magma`'s algorithms is uncertain, an advantage they certainly have over `Magma` is that they are free and open source.

The use of interval arithmetic would improve our handling of real numbers. Currently, we implement real numbers with the `mpfr` library. Interval arithmetic would be superior as it would let us test whether the precision used is insufficient for a correct result in our functions using real numbers. It also would give us the precision of the final result. An efficient implementation of interval arithmetic is part of the `Arb` library which has recently been merged into the library `Flint`.

Our experiments with alternative approaches to linear algebra in were discussed in Section 5.2.4. We observed that it is common to arrive in the situation where the relations in the log unit lattice give us a small multiple of the regulator. From here generating additional relations to obtain the true regulator can be expensive. Saturation of the unit group, a technique which Biasse-Fieker [BF12] found to be effective, appears to be a less costly alternative. Although some experimentation would be needed to determine the most effective way to integrate saturation of the unit group, we think it would be worthwhile. Saturation of the unit group is a separate technique from saturation of the class group which we found much less effective. In Section 5.2.4 we also considered an alternative approach to constructing relations in the log unit lattice. An investigation into this approach is also worth pursuing.

The efficiency of our implementation could be increased with further work on parameter tuning. Specifically, an extensive search of the parameter space could be carried out. The use of dedicated hardware would allow conducting many runs in parallel which would make this search easier and enable incorporating data from number fields of larger discriminant size. In some computer science disciplines, advanced tools have been developed to automate much of the parameter tuning process. However, we are not aware of any such tools specifically designed for automating parameter tuning in computational number theory algorithms.

Another potential direction for research is analyzing the time complexity of our self-initialization approach. The time complexity of self-initialization approaches in quadratic number fields has been analyzed (see for example [Kle16]). Our approach would also benefit from such analysis. Beyond its theoretical significance, an analysis of our approach's time complexity could also lead to practical improvements.

In considering the broader impact of our research, several avenues emerge for further exploration. A promising one is the potential application of our work to isogeny-based cryptographic protocols for genus two hyperelliptic curves. Additionally, the applications to computational number theory outlined in Section 1.2, including solving Diophantine equations and tabulation of class groups and unit groups, are also interesting. Our self-initialization approach can be used for these problems, particularly for applications where the discriminant of the quartic field is large. Given the improved efficiency of our approach when the roots of the defining polynomial are small, an exciting direction would be to identify applications where such a defining polynomial is known.

Bibliography

[All21] Bill Allombert. PARI developer mailing list, 2021. https://pari.math.u-bordeaux.fr/archives/pari-dev-2106/msg00002.html.

[Bac90] Eric Bach. Explicit bounds for primality testing and related problems. *Mathematics of Computation*, 55(191):355–380, 1990.

[Bac95] Eric Bach. Improved approximations for Euler products. In *Number Theory, CMS Conference Proceedings*, volume 15, pages 13–28, 1995.

[BBdV$^+$17] Jens Bauch, Daniel Bernstein, Henry de Valence, Tanja Lange, and Christine van Vredendaal. Short generators without quantum computers: The case of multiquadratics. In Jean-Sébastien Coron and Jesper Buus Nielsen, editors, *Advances in Cryptology - EUROCRYPT 2017*, volume 10210 of *Lecture Notes in Computer Science*, pages 27–59, 2017.

[BC18] Karim Belabas and Henri Cohen. Modular forms in Pari/GP. *Research in the Mathematical Sciences*, 5:1–19, 2018.

[BCP97] Wieb Bosma, John Cannon, and Catherine Playoust. The Magma algebra system. I. The user language. *Journal of Symbolic Computation*, 24(3-4):235–265, 1997.

[BD91] Johannes Buchmann and Stephan Düllmann. A probabilistic class group and regulator algorithm and its implementation. In *Computational Number Theory*, pages 53–72. De Gruyter, 1991.

[BDyDF08] Karim Belabas, Francisco Diaz y Diaz, and Eduardo Friedman. Small generators of the ideal class group. *Mathematics of Computation*, 77(262):1185–1197, 2008.

[Bel04] Karim Belabas. Topics in computational algebraic number theory. *Journal de Théorie des Nombres de Bordeaux*, 16(1):19–63, 2004.

[Ber04] Daniel Bernstein. *How To Find Smooth Parts Of Integers*. 2004. Available at http://cr.yp.to/factorization/smoothparts-20040510.pdfll.

[BF12] Jean-François Biasse and Claus Fieker. Improved techniques for computing the ideal class group and a system of fundamental units in number fields. In *Algorithmic Number Theory, 10th International Symposium, ANTS-IX, San Diego CA, USA, July 9-13, 2012. Proceedings*, volume 1 of *Open Book Series*, pages 113–133. Mathematical Science Publishers, 2012.

[BF14] Jean-François Biasse and Claus Fieker. Subexponential class group and unit group computation in large degree number fields. *LMS Journal of Computation and Mathematics*, 17(A):385–403, 2014.

[BFP93] Johannes Buchmann, David Ford, and Michael Pohst. Enumeration of quartic fields of small discriminant. *Mathematics of Computation*, 61(204):873–879, 1993.

[BGG$^+$20] Fabrice Boudot, Pierrick Gaudry, Aurore Guillevic, Nadia Heninger, Emmanuel Thomé, and Paul Zimmermann. Comparing the difficulty of factorization and discrete logarithm: a 240-digit experiment. In *Advances in Cryptology–CRYPTO 2020: 40th Annual International Cryptology Conference, CRYPTO 2020, Santa Barbara, CA, USA, August 17–21, 2020, Proceedings, Part II 40*, pages 62–91. Springer, 2020.

[Bia10] Jean-François Biasse. Improvements in the computation of ideal class groups of imaginary quadratic number fields. *Advances in Mathematics of Communications*, 4(2):141–154, 2010.

[Bia14a] Jean-François Biasse. An $L(1/3)$ algorithm for ideal class group and regulator computation in certain number fields. *Mathematics of Computation*, 83(288):2005–2031, 2014.

[Bia14b] Jean-François Biasse. Subexponential time relations in the class group of large degree number fields. *Advances in Mathematics of Communications*, 8(4):407, 2014.

[Bia23] Jean-François Biasse. Personal communication. Email, Dec 2023.

[BJ10] Jean-François Biasse and Michael J. Jacobson Jr. Practical improvements to class group and regulator computation of real quadratic fields. In Guillaume Hanrot, François Morain, and Emmanuel Thomé, editors, *Algorithmic Number Theory, 9th International Symposium, ANTS-IX, Nancy, France, July 19-23, 2010. Proceedings*, volume 6197 of *Lecture Notes in Computer Science*, pages 50–65. Springer, 2010.

[BJN$^+$99] Johannes Buchmann, Michael Jacobson, Jr., Stefan Neis, Patrick Theobald, and Damian Weber. Sieving methods for class group computation. In *Algorithmic Algebra and Number Theory*, pages 3–10. Springer, 1999.

[BJS10] Jean-François Biasse, Michael Jacobson, Jr., and Alan K. Silvester. Security estimates for quadratic field based cryptosystems. In Ron Steinfeld and Philip Hawkes, editors, *Information Security and Privacy - 15th Australasian Conference, ACISP 2010, Sydney, Australia, July 5-7, 2010. Proceedings*, volume 6168 of *Lecture Notes in Computer Science*, pages 233–247. Springer, 2010.

[BKV19] Ward Beullens, Thorsten Kleinjung, and Frederik Vercauteren. CSI-FiSh: Efficient isogeny based signatures through class group computations. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 227–247. Springer, 2019.

[BLP93] Joe Buhler, Hendrik Lenstra, and Carl Pomerance. Factoring integers with the number field sieve. In *The development of the number field sieve*, pages 50–94. Springer, 1993.

[BPvS94] Johannes Buchmann, Michael Pohst, and Graf von Schmettow. On unit groups and class groups of quartic fields of signature $(2, 1)$. *Mathematics of Computation*, 62(205):387–390, 1994.

[BS16] Jean-François Biasse and Fang Song. Efficient quantum algorithms for computing class groups and solving the principal ideal problem in arbitrary degree number fields. In Robert Krauthgamer, editor, *Proceedings of the Twenty-Seventh Annual ACM-SIAM Symposium on Discrete Algorithms, SODA 2016, Arlington, VA, USA, January 10-12, 2016*, pages 893–902. SIAM, 2016.

[Buc90] Johannes Buchmann. A subexponential algorithm for the determination of class groups and regulators of algebraic number fields. In Catherine Goldstein, editor, *Séminaire de Théorie des Nombres, Paris 1988–1989*, pages 27–41, Boston, 1990. Birkhauser.

[BvV19] Jean-François Biasse and Christine van Vrendendaal. Improved techniques for computing the ideal class group and a system of fundamental units in number fields. In *Algorithmic Number Theory, 13th International Symposium, ANTS-XIII, Madison WI, USA, July 16-20, 2018. Proceedings*, volume 2 of *Open Book Series*, pages 103–118. Mathematical Science Publishers, 2019.

[BW90] Johannes Buchmann and Hugh Williams. Quadratic fields and cryptography. *Number Theory and Cryptography*, 154:9–25, 1990.

[Cau91] A.L. Cauchy. *Oeuvres Completes: Series 2. : Vol.: 9 : Exercices de Mathématiques : (Anciens Exercices)*. Gauthier-Villars et Fils, Imprimeurs- Libraire, 1891.

[Cav02] Stefania Cavallar. *Strategies in filtering in the number field sieve*. PhD thesis, Leiden University, 2002.

[CBFS23a] John Cannon, Wieb Bosma, Claus Fieker, and Allan Steel. *Handbook of Magma functions*. 2.27.6 edition, 2023.

[CBFS23b] John Cannon, Wieb Bosma, Claus Fieker, and Allan Steel. Quadratic class groups. In *Handbook of Magma functions*. 2.27.6 edition, 2023. https://magma.maths.usyd.edu.au/magma/handbook/text/393 [Online; accessed 30 January 2024].

[CD91] Henri Cohen and Diaz Y Diaz. A polynomial reduction algorithm. *Journal de théorie des nombres de Bordeaux*, 3(2):351–360, 1991.

[CDO97] Henri Cohen, F Diaz Y Diaz, and Michel Olivier. Subexponential algorithms for class group and unit computations. *Journal of Symbolic Computation*, 24(3-4):433–441, 1997.

[CLM$^+$18] Wouter Castryck, Tanja Lange, Chloe Martindale, Lorenz Panny, and Joost Renes. CSIDH: an efficient post-quantum commutative group action. In *International Conference on the Theory and Application of Cryptology and Information Security*, pages 395–427. Springer, 2018.

[CM87] Henri Cohen and Jacques Martinet. Class groups of number fields: numerical heuristics. *Mathematics of Computation*, 48(177):123–137, 1987.

[Coh93] Henri Cohen. *A Course in Computational Algebraic Number Theory*, volume 138 of *Graduate Texts in Mathematics*. Springer-Verlag, Berlin, 3 edition, 1993.

[Con97] Scott Patrick Contini. *Factoring integers with the self-initializing quadratic sieve*. PhD thesis, University of Georgia, 1997.

[CS05] Zhuliang Chen and Arne Storjohann. A BLAS based C library for exact linear algebra on integer matrices. In *Proceedings of the 2005 international symposium on Symbolic and algebraic computation*, pages 92–99, 2005.

[CS20] Craig Costello and Benjamin Smith. The supersingular isogeny problem in genus 2 and beyond. In *PQCrypto 2020-11th International Conference on Post-Quantum Cryptography*, Lecture Notes in Computer Science, pages 151–168. Springer, 2020.

[DER86] Iain S Duff, Al M Erisman, and John K Reid. *Direct methods for sparse matrices*. Monographs on Numerical Analysis. Oxford University Press, 1986.

[DH84] James A Davis and Diane B Holdridge. Factorization using the quadratic sieve algorithm. In *Advances in Cryptology: Proceedings of Crypto 83*, pages 103–113. Springer, 1984.

[Dül91] Stephan Düllmann. *Ein Algorithmus zur Bestimmung der Klassengruppe positiv definiter binärer quadratischer Formen*. PhD thesis, Universität des Saarlandes, Saarbrücken, 1991.

[FK05] Jens Franke and Thorsten Kleinjung. Continued fractions and lattice sieving. *Special-Purpose Hardware for Attacking Cryptographic Systems–SHARCS*, page 40, 2005.

[FP85] Ulrich Fincke and Michael Pohst. Improved methods for calculating vectors of short length in a lattice, including a complexity analysis. *Mathematics of Computation*, 44(170):463–471, 1985.

[Fun84] Takeo Funakura. On integral bases of pure quartic fields. *Mathematical Journal of Okayama University*, 26(1):27–41, 1984.

[Gél17] Alexandre Gélin. *Class group computations in number fields and applications to cryptology*. PhD thesis, L'université Pierre Et Marie Curie, 2017.

[Gél18a] Alexandre Gélin. On the complexity of class group computations for large degree number fields. *CoRR*, abs/1810.11396, 2018.

[Gél18b] Alexandre Gélin. Reducing the complexity for class group computations using small defining polynomials. *CoRR*, abs/1810.12010, 2018.

[GJ16] Alexandre Gélin and Antoine Joux. Reducing number field defining polynomials: An application to class group computations. *LMS Journal of Computation and Mathematics*, 19(A):315–331, 2016.

[GJS01] Mark Giesbrecht, Michael Jacobson, Jr., and Arne Storjohann. Algorithms for large integer matrix problems. In Serdar Boztaş and Igor E. Shparlinski, editors, *Applied Algebra, Algebraic Algorithms and Error-Correcting Codes*, pages 297–307, Berlin, Heidelberg, 2001. Springer Berlin Heidelberg.

[GM22] Loïc Grenié and Giuseppe Molteni. Breaking the 4 barrier for the bound of a generating set of the class group, 2022. arXiv:2212.09461 [math].

[Gra08] Andrew Granville. Smooth numbers: computational number theory and beyond. *Algorithmic number theory: lattice, number fields, curves and cryptography*, 44:267–323, 2008.

[HM89] James Hafner and Kevin McCurley. A rigorous subexponential algorithm for computation of class groups. *Journal of the American Mathematical Society*, 2(4):837–850, 1989.

[HM97] George Havas and Bohdan S Majewski. Integer matrix diagonalization. *Journal of Symbolic Computation*, 24(3-4):399–408, 1997.

[HSW95] James G Huard, Blair Kenneth Spearman, and Kenneth S Williams. Integral bases for quartic fields with quadratic subfields. *Journal of number theory*, 51(1):87–102, 1995.

[Jac99a] Michael Jacobson, Jr. Applying sieving to the computation of quadratic class groups. *Mathematics of Computation*, 68(226):859–867, 1999.

[Jac99b] Michael Jacobson, Jr. *Subexponential class group computation in quadratic orders*. PhD thesis, Technische Universität Darmstadt, Darmstadt, Germany, 1999.

[JJWW04] Michael J Jacobson Jr., Hugh C Williams, and Kjell Wooding. Imaginary cyclic quartic fields with large minus class numbers. In *International Algorithmic Number Theory Symposium*, pages 280–292. Springer, 2004.

[Jou09] Antoine Joux. *Algorithmic Cryptanalysis*. Cryptography and Network Security. Chapman & Hall / CRC Press, 2009.

[JW09] Michael Jacobson, Jr. and Hugh Williams. *Solving the Pell Equation*. CMS Books in Mathematics. Springer, 2009.

[KAF$^+$10] Thorsten Kleinjung, Kazumaro Aoki, Jens Franke, Arjen K Lenstra, Emmanuel Thomé, Joppe W Bos, Pierrick Gaudry, Alexander Kruppa, Peter L Montgomery, and Dag Arne Osvik. Factorization of a 768-bit RSA modulus. In *Advances in Cryptology–CRYPTO 2010: 30th Annual Cryptology Conference, Santa Barbara, CA, USA, August 15-19, 2010. Proceedings 30*, pages 333–350. Springer, 2010.

[Kan87] Ravi Kannan. Minkowski's convex body theorem and integer programming. *Mathematics of Operations Research*, 12(3):415–440, 1987.

[Kle16] Thorsten Kleinjung. Quadratic sieving. *Mathematics of Computation*, 85(300):1861–1873, 2016.

[Kur50] Sigekatu Kuroda. Über die Klassenzahlen algebraischer Zahlkörper. *Nagoya Mathematical Journal*, 1:383–406, 1950.

[KW89] Luise-Charlotte Kappe and Bette Warren. An elementary test for the Galois group of a quartic polynomial. *The American Mathematical Monthly*, 96(2):133–137, 1989.

[Laz89] Andrew J. Lazarus. *The class number and cyclotomy of simplest quartic fields.* PhD thesis, University of California, Berkeley, 1989.

[Lem94] Franz Lemmermeyer. Kuroda's class number formula. *Acta Arithmetica*, 66(3):245–260, 1994.

[Len17] Arjen K. Lenstra. General purpose integer factoring. *IACR Cryptol. ePrint Arch.*, page 1087, 2017.

[LHLL82] Arjen Lenstra, Jr. Hendrik Lenstra, and László Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982.

[LMF24a] The LMFDB Collaboration. Canonical defining polynomial for number fields. https://www.lmfdb.org/knowledge/show/nf.polredabs, 2024. [Online; accessed 30 January 2024].

[LMF24b] The LMFDB Collaboration. The L-functions and modular forms database. https://www.lmfdb.org, 2024. [Online; accessed 30 January 2024].

[LO91] Brian A LaMacchia and Andrew M Odlyzko. Solving large sparse linear systems over finite fields. In *Advances in Cryptology-CRYPTO'90: Proceedings 10*, pages 109–133. Springer, 1991.

[Luo24] Qinglong Luo. *Computing Class Groups of Cubic Orders Using Kleinjung's Sieving Method*. PhD thesis, University of Calgary, 2024.

[Mar24] David Marquis. Github, 2024. https://github.com/davidamarquis/quarticcg.

[MG94] Maurice Mignotte and Philippe Glesser. Landau's inequality via Hadamard's. *Journal of symbolic computation*, 18(4):379–383, 1994.

[MJ16] Anton Mosunov and Michael Jacobson, Jr. Unconditional class group tabulation of imaginary quadratic fields to $|\Delta < 2^{40}|$. *Mathematics of Computation*, 85(300):1983–2009, 2016.

[MS23] Madhurima Mukhopadhyay and Palash Sarkar. Pseudo-random walk on ideals: practical speed-up in relation collection for class group computation. In *International Symposium on Cyber Security, Cryptology, and Machine Learning*, pages 18–31. Springer, 2023.

[MW01a] Daniele Micciancio and Bogdan Warinschi. A linear space algorithm for computing the Hermite normal form. In *Proceedings of the 2001 international symposium on Symbolic and algebraic computation*, pages 231–236, 2001.

[MW01b] Daniele Micciancio and Bogdan Warinschi. A linear space algorithm for computing the hermite normal form. In *Proceedings of the 2001 international symposium on Symbolic and algebraic computation*, pages 231–236, 2001.

[Nei02] Stefan Neis. *Zur Berechnung von Klassengruppen.* PhD thesis, Technische Universität Darmstadt, 2002.

[Neu99] Jürgen Neukirch. *Algebraic number theory*, volume 322 of *Grundlehren der Mathematischen Wissenschaften [Fundamental Principles of Mathematical Sciences].* Springer-Verlag, Berlin, 1999. Translated by Norbert Schappacher.

[Pau96] Sachar Paulus. An algorithm of subexponential type computing the class group of quadratic orders over principal ideal domains. In *Algorithmic Number Theory – ANTS-II*, volume 1122 of *Lecture Notes in Computer Science*, pages 243–257. Springer, 1996.

[Pom96] Carl Pomerance. A tale of two sieves. *Notices of the AMS*, 85:175, 1996.

[PS10] Clément Pernet and William Stein. Fast computation of hermite normal forms of random integer matrices. *Journal of Number Theory*, 130(7):1675–1683, 2010.

[PS13] Colton Pauderis and Arne Storjohann. Computing the invariant structure of integer matrices: fast algorithms into practice. In *Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation*, pages 307–314, 2013.

[PST88] Carl Pomerance, Jeffrey W Smith, and Randy Tuler. A pipeline architecture for factoring large integers with the quadratic sieve algorithm. *SIAM Journal on Computing*, 17(2):387–403, 1988.

[PZ79] Michael Pohst and Hans Zassenhaus. On unit computation in real quadratic fields. In *International Symposium on Symbolic and Algebraic Manipulation*, pages 140–152. Springer, 1979.

[PZ85] Michael Pohst and Hans Zassenhaus. Über die Berechnung von Klassenzahlen und Klassengruppen algebraischer Zahlkörper. *Journal Für Die Reine Und Angewandte Mathematik*, 361:50–72, 1985.

[PZ89] Michael Pohst and Hans Zassenhaus. *Algorithmic algebraic number theory.* Cambridge University Press, 1989.

[Sey87] Martin Seysen. A probabilistic factorization algorithm with quadratic forms of negative discriminant. *Mathematics of Computation*, 48(178):757–780, 1987.

[Sha74] Daniel Shanks. The simplest cubic fields. *Mathematics of computation*, 28(128):1137–1152, 1974.

[Sil87] Robert D Silverman. The multiple polynomial quadratic sieve. *Mathematics of Computation*, 48(177):329–339, 1987.

[SL96] Peter Stevenhagen and Jr. Hendrik W Lenstra. Chebotarëv and his density theorem. *The Mathematical Intelligencer*, 18:26–37, 1996.

[Tea23a] The CADO-NFS Development Team. CADO-NFS, an implementation of the number field sieve algorithm, 2023. development version.

[tea23b] The FLINT team. *FLINT: Fast Library for Number Theory*, 2023. Version 3.0.0, https://flintlib.org.

[The] The PARI Group, University Bordeaux. *PARI/GP Documentation.* [Online; accessed 30 January 2024].

[The20] The PARI Group, University Bordeaux. *PARI/GP version* `2.11.2`, 2020. Available at https://pari.math.u-bordeaux.fr/.

[The23] The Sage Developers. *SageMath, the Sage Mathematics Software System (Version 10.0.0)*, 2023. https://www.sagemath.org.

[Thi95] Christoph Thiel. Short proofs using compact representations of algebraic integers. *Journal of Complexity*, 11(3):310–329, 1995.

[Tho22] Emmanuel Thomé. Lecture notes on the number field sieve (cse-291 winter 2022), 2022. Available at https://members.loria.fr/EThome/teaching/2022-cse-291-14/.

[Vol00] Ulrich Vollmer. Asymptotically fast discrete logarithms in quadratic number fields. In Wieb Bosma, editor, *Algorithmic Number Theory – ANTS-IV*, Lecture Notes in Computer Science, pages 581–594, Berlin, Heidelberg, 2000. Springer Berlin Heidelberg.

Appendix A

Appendix

In order to compare our approach in self-initialization implementation to Magma we need an algorithm that is close to ours in terms of inputs and outputs. The following Magma algorithm computes the class and unit groups. We use this to benchmark against the self-initialization implementation.

Algorithm 4 Magma Compute Class Group and Unit Group

```
Input: $f$ : defining polynomial of the number field
Output: Class group and unit group
1: ClassAndUnitGroup:=function($f$)
2:     $O$:=MaximalOrder($f$);
3:     cg,cgmap:=ClassGroup($O$: Proof:="GRH");
4:     ug,ugmap:=UnitGroup($O$: GRH:=true);
5:     return cg, ug;
6: end function;
```

This algorithm is called by first constructing a polynomial ring over the integers and then calling Magma's implementation with a polynomial argument. The documentation for ClassGroup explains the "GRH" argument for the Proof flag:

> "a bound based on the GRH is used to replace the Minkowski bound...The result will hence be correct under the GRH."

The GRH flag in UnitGroup has a similar purpose.

Although we cannot know the details of `Magma`'s class group algorithm, a small number of parameters used by `Magma`'s implementation are clearly identified in the logs of `Magma`'s implementation. The logs give us the smoothness bound and large prime bound as these are relevant to our algorithm. From the logs we can also identify three stages of the algorithm's running time: initialization, time until the relation matrix reaches full rank, the time for linear algebra, and the total time.