Pseudo-Random Walk on Ideals: Practical Speed-Up in Relation Collection for Class Group Computation

Madhurima Mukhopadhyay and Palash Sarkar

Applied Statistics Unit
Indian Statistical Institute
203 B. T. Road
Kolkata
India 700108
{madhurima_r,palash}@isical.ac.in

June 11, 2021

Abstract

We introduce a technique to obtain practical speed up for relation collection in class group computations. The idea is to perform a pseudo-random walk over the ideals. The ideals visited by the walk are used in the manner exactly as in the previous algorithm due to Gélin (2018). Under the heuristic assumption that the ideals visited by the walk behave as the ideals randomly generated in Gélin's algorithm, the asymptotic complexity of the new algorithm remains the same as that of Gélin's algorithm. The main advantage of the new method over Gélin's method is that the pseudo-random walk requires a single ideal multiplication to generate the next ideal in the walk, whereas Gélin's algorithm requires a number of ideal multiplications to generate each ideal to be tested. We have made Magma implementations of both the new algorithm and Gélin's algorithm. Timing results confirm that there is indeed a substantial practical speed-up in relation collection by the new algorithm over Gélin's algorithm.

Keywords: class group, pseudo-random walk.
MSC (2010): 11Y40, 94A60.

1 Introduction

A basic problem in computational algebraic number theory is to compute the class group of the ring of integers of a number field and more generally, the class group of an order of the ring of integers. The complexity of the computational problem is expressed in terms of the discriminant $\Delta_{\mathcal{K}}$ of the number field $\mathcal{K}$. Presently, there is no polynomial time (in $\log|\Delta_{\mathcal{K}}|$) algorithm for computing the class group. Shanks [15, 16] had proposed the first algorithm for computing the class group in time $O(|\Delta_{\mathcal{K}}|^{1/5})$ assuming that the generalised Riemann hypothesis (GRH) holds.

Later improvements have provided sub-exponential algorithms for the problem, where for $N > 0$, $\alpha \in [0,1]$ and $c \geq 0$, the sub-exponential expression $L_N(\alpha, c)$ is defined as follows.

$$L_N(\alpha, c) \quad = \quad \exp\left((c + o(1))(\log N)^{\alpha}(\log\log N)^{1-\alpha}\right). \tag{1}$$

Hafner and McCurley [13] proposed the first sub-exponential algorithm for computing the class group of imaginary quadratic number fields also assuming GRH. The runtime of their algorithm is $L_{|\Delta_{\mathcal{K}}|}(1/2, \sqrt{2})$. For general number fields, Buchmann [5] extended the Hafner-McCurly algorithm to obtain a sub-exponential algorithm when the extension degree is fixed. Several ideas which improve the implementation of Buchmann's algorithm were introduced by Cohen, Diaz Y Diaz and Olivier [8]. Biasse and Fieker [3] described an algorithm to compute the class group of any number field in time $L_{|\Delta_{\mathcal{K}}|}(2/3+\epsilon, c)$, for arbitrarily small ϵ and $c > 0$ without any restriction on the extension degree. This algorithm was simplified by Gélin [9] and the asymptotic complexity was improved. It is to be noted that the complexity analyses of all the algorithms are heuristic; one of the main heuristic assumptions being that smoothness results for general ideals are assumed to apply to the restricted set of ideals considered by the algorithms.

Sub-exponential class group computation algorithms are index calculus algorithms having two dominant steps, namely the relation collection step and the linear algebra step. The linear algebra step essentially consists of computing the Smith Normal Form of a matrix which is the output of the relation collection step. Progress in the class group computation algorithms has mainly been in the relation collection step.

In the present work, we provide a new heuristic algorithm to improve the practical runtime of the relation collection step in Gélin's algorithm [9]. The core idea of previous ideal reduction algorithms is to randomly generate ideals on which lattice techniques are used to obtain principal ideals of bounded norms which can then be tested for smoothness over a pre-defined factor basis. Presently, the most simplified form of this algorithm is due to Gélin [9]. The random generation of the ideals in Gélin's algorithm requires exponentiating some randomly chosen prime ideals in the factor basis and then multiplying the resulting ideals together. This requires performing a number of ideal multiplications.

We propose to perform a pseudo-random walk on ideals. Once the initial ideal has been computed, each step of the walk requires a single ideal multiplication. The ideals generated in each step can be processed using lattice techniques as in Gélin's algorithm. The parameters which determine the asymptotic complexity of Gélin's algorithm [9] are not changed. So, under the heuristic assumption that the ideals in the pseudo-random walk behave like random ideals, the asymptotic complexity of the new algorithm remains unchanged from that of Gélin's algorithm. The improvement is a practical gain in the time to generate a relation.

We have made a Magma implementation of both the new algorithm and Gélin's algorithm [9]. Since our goal is to compare the new algorithm with Gélin's algorithm [9], we did not try to compute the class group for one particular field. Instead, we chose four number fields and conducted experiments with both the algorithms to collect a number of relations. The timing results obtained from these experiments indeed confirm that the speed improvement of the new algorithm over Gélin's algorithm [9].

Related Work

A related line of work considers algorithms with improved complexity when the coefficients of the generating polynomial of the number field are small. Biasse [2] and Biasse and Fieker [3] showed an $L_{|\Delta_{\mathcal{K}}|}(a,\cdot)$ algorithm for certain fields where a can be as low as $1/3$. Gélin and Joux [11] extended the result of Biasse and Fieker [3] to a larger class of fields by describing an algorithm for obtaining polynomials with small coefficients for certain fields. The most recent work in this line is the sieving based relation collection algorithm due to Gélin [10].

The set of relations obtained for computing the class group are also used for computing the regulator and a set of generators of the unit group. In fact, the regulator and the class number (i.e., the order of the class group) are required to be computed together. The techniques for decomposing an ideal over a factor basis are also used to obtain algorithms for the principal ideal problem. We refer to [7] for details.

Cryptographic Applications

As discussed above, there is no polynomial time algorithm to compute the class group and the best known algorithms have sub-exponential complexity. So, for a number field with a sufficiently large discriminant, the order of the class group can be considered to be unknown. Such a hidden order group forms the basis for several cryptographic primitives [4, 6]. Concrete suggestions for instantiating these primitives have been made using class groups of imaginary quadratic fields. In principal, though, the class group of a general number field can also be used. The security versus efficiency question of using class groups of general number fields versus those of imaginary quadratic fields remain to be studied. Progress in algorithms for computing class groups influences the choice of number fields over which the relevant cryptographic primitives can be securely implemented.

2 Preliminaries

Let $\mathbb{Q}$, $\mathbb{R}$ and $\mathbb{C}$ respectively denote the fields of rational, real and complex numbers. The ring of integers will be denoted by $\mathbb{Z}$. In the following we provide a brief overview of some relevant facts regarding the problem of computing the class group. Further details can be found in [7].

An algebraic number field is a finite extension of $\mathbb{Q}$. More concretely, $\mathcal{K}$ is an algebraic number field defined by $T(X)$ if $\mathcal{K} \simeq \mathbb{Q}[X]/(T(X))$ for some irreducible $T(X) \in \mathbb{Z}[X]$. The extension degree of $\mathcal{K}$ over $\mathbb{Q}$ is equal to the degree of $T(X)$.

Suppose $T(X)$ is of degree n. Then $T(X)$ has n roots in $\mathbb{C}$. Since the complex roots occurs in pairs, let r_1 be the number of real roots of $T(X)$ and $2r_2$ be the number of complex roots of $T(X)$, where $r_1 + 2r_2 = n$. Suppose $\xi_1, \ldots, \xi_n$ be the roots of $T(X)$. For $i = 1, \ldots, n$, the embedding σ_i of $\mathcal{K}$ into $\mathbb{C}$ is defined in the following manner. For $a_0, \ldots, a_{n-1} \in \mathbb{Q}$,

$$\sigma_i(a_0 + a_1 X + \cdots + a_{n-1}X^{n-1}) \mapsto a_0 + a_1\xi_i + \cdots + a_{n-1}\xi_i^{n-1}.$$

So $\sigma_1, \ldots, \sigma_n$ are n embeddings of $\mathcal{K}$ into $\mathbb{C}$. The embedding σ_i is called real or complex according as ξ_i is real or complex.

The norm and trace functions $\mathcal{N}_{\mathcal{K}/\mathbb{Q}}$ and $\mathcal{T}_{\mathcal{K}/\mathbb{Q}}$ respectively from $\mathcal{K}$ to $\mathbb{Q}$ are defined as follows. For $\alpha \in \mathcal{K}$

$$\mathcal{N}_{\mathcal{K}/\mathbb{Q}}(\alpha) = \prod_{i=1}^{n} \sigma_i(\alpha), \qquad \mathcal{T}_{\mathcal{K}/\mathbb{Q}}(\alpha) = \sum_{i=1}^{n} \sigma_i(\alpha).$$

The norm and trace functions are respectively multiplicative and additive on $\mathcal{K}$. For computational purposes, it is important to be able to compute the norm of an element. Any element $\alpha \in \mathcal{K}$ can be written as $\alpha = A(X)/d$ where $d \in \mathbb{Z}$ and $A(X) \in \mathbb{Z}[X]$. From Proposition 4.3.4 of [7], we have

$$\mathcal{N}_{\mathcal{K}/\mathbb{Q}}(\alpha) = \frac{1}{d^n} \mathrm{Res}\,(T(X), A(X)) \tag{2}$$

where Res denotes the resultant.

An element $\alpha \in \mathcal{K}$ is said to be an integral element of $\mathcal{K}$, if it satisfies a monic polynomial with integer coefficients. The set of all integral elements of $\mathcal{K}$ is denoted by $\mathcal{O}_{\mathcal{K}}$. The set $\mathcal{O}_{\mathcal{K}}$ is a ring and is also an n-dimensional $\mathbb{Z}$-module (i.e., a lattice). A $\mathbb{Z}$-basis of $\mathcal{O}_{\mathcal{K}}$ is called an integral basis of $\mathcal{K}$. The field of fractions of $\mathcal{O}_{\mathcal{K}}$ is $\mathcal{K}$. For any $\alpha \in \mathcal{K}$, there is an integer a, such that $a\alpha \in \mathcal{O}_{\mathcal{K}}$.

Let $b_1, \ldots, b_n$ be an integral basis of $\mathcal{K}$. The discriminant $\Delta_{\mathcal{K}}$ of the number field $\mathcal{K}$ is defined to be

$$\Delta_{\mathcal{K}} \quad = \quad (\det(\sigma_i(b_j)))^2 = \det\left(\mathcal{T}_{\mathcal{K}/\mathbb{Q}}(b_i b_j)\right).$$

A fractional ideal $\mathfrak{a}$ of $\mathcal{O}_{\mathcal{K}}$ is an $\mathcal{O}_{\mathcal{K}}$-submodule of $\mathcal{K}$ for which there is a $d \in \mathcal{O}_{\mathcal{K}}$ such that $d\mathfrak{a} \subseteq \mathcal{O}_{\mathcal{K}}$. The element d is said to be a denominator of $\mathfrak{a}$. It is easy to argue that $d\mathfrak{a}$ is an ideal of $\mathcal{O}_{\mathcal{K}}$. An (ordinary) ideal of $\mathcal{O}_{\mathcal{K}}$ is a fractional ideal (taking $d = 1$) and is called an integral ideal. If $\mathfrak{a}$ and $\mathfrak{b}$ are fractional ideals, then the product $\mathfrak{a}\mathfrak{b}$ is defined in exactly the same manner as for integral ideals. Any fractional ideal $\mathfrak{a}$ of $\mathcal{O}_{\mathcal{K}}$ is invertible, i.e., there is a fractional ideal $\mathfrak{b}$, such that $\mathfrak{a}\mathfrak{b} = \mathcal{O}_{\mathcal{K}}$. The inverse of $\mathfrak{a}$ is $\mathfrak{a}^{-1} = \{\alpha \in \mathcal{K} : \alpha\mathfrak{a} \subseteq \mathcal{O}_{\mathcal{K}}\}$. Any fractional idea $\mathfrak{a}$ can be written as $\mathfrak{b}/\mathfrak{c}$, where $\mathfrak{b}$ and $\mathfrak{c}$ are integral ideals; in fact, one may choose $\mathfrak{c} = d\mathcal{O}_{\mathcal{K}}$ where d is a denominator of $\mathfrak{a}$.

Let $\mathcal{I}_{\mathcal{O}_{\mathcal{K}}}$ be the set of all fractional ideals of $\mathcal{O}_{\mathcal{K}}$. Then $\mathcal{I}_{\mathcal{O}_{\mathcal{K}}}$ is a commutative group with $\mathcal{O}_{\mathcal{K}}$ as the identity element. For $\alpha \in \mathcal{K}$, the set $(\alpha) = \alpha\mathcal{O}_{\mathcal{K}}$ is said to be a principal fractional ideal of $\mathcal{O}_{\mathcal{K}}$. Let $\mathcal{P}_{\mathcal{O}_{\mathcal{K}}}$ be the set of all principal fractional ideals of $\mathcal{O}_{\mathcal{K}}$. Then $\mathcal{P}_{\mathcal{O}_{\mathcal{K}}}$ is a subgroup of $\mathcal{I}_{\mathcal{O}_{\mathcal{K}}}$. The quotient group $\mathrm{Cl}_{\mathcal{O}_{\mathcal{K}}} = \mathcal{I}_{\mathcal{O}_{\mathcal{K}}}/\mathcal{P}_{\mathcal{O}_{\mathcal{K}}}$ is said to be the ideal class group of $\mathcal{O}_{\mathcal{K}}$. It is known that $\mathrm{Cl}_{\mathcal{O}_{\mathcal{K}}}$ is finite and the order of $\mathrm{Cl}_{\mathcal{O}_{\mathcal{K}}}$ is said to be the class number of $\mathcal{K}$.

Let $\mathfrak{a}$ be an integral ideal of $\mathcal{O}_{\mathcal{K}}$. It can be proved that the quotient group $\mathcal{O}_{\mathcal{K}}/\mathfrak{a}$ is finite. The norm of $\mathfrak{a}$, written as $\mathcal{N}(\mathfrak{a})$ is defined to be the cardinality of $\mathcal{O}_{\mathcal{K}}/\mathfrak{a}$. If $\mathfrak{a}$ is a principal ideal $\alpha\mathcal{O}_{\mathcal{K}}$, then it can be proved that $\mathcal{N}(\mathfrak{a}) = |\mathcal{N}(\alpha)|$. The notion of norm can be extended to fractional ideals. Suppose $\mathfrak{a}$ is a fractional ideal which is written as $\mathfrak{a} = \mathfrak{b}/\mathfrak{c}$, where $\mathfrak{b}$ and $\mathfrak{c}$ are integral ideals. Then $\mathcal{N}(\mathfrak{a}) = \mathcal{N}(\mathfrak{b})/\mathcal{N}(\mathfrak{c})$. A fractional ideal $\mathfrak{a}$ can be written as the ratio of two integral ideals $\mathfrak{b}$ and $\mathfrak{c}$ in more than one way. The definition of the norm of $\mathfrak{a}$ does not depend upon the choice of $\mathfrak{b}$ and $\mathfrak{c}$.

Unique factorisation holds for the fractional ideals of $\mathcal{O}_{\mathcal{K}}$. If $\mathfrak{a}$ is a fractional ideal, then there are unique prime ideals $\mathfrak{p}_1, \ldots, \mathfrak{p}_k$ and non-zero integers $e_1, \ldots, e_k$ such that $\mathfrak{a} = \mathfrak{p}_1^{e_1} \cdots \mathfrak{p}_k^{e_k}$.

A basic computational problem in algebraic number theory is to compute the class group. Let K be a number field and $\mathcal{B}$ be a set of ideals of $\mathcal{O}_{\mathcal{K}}$ such that any $[\mathfrak{a}] \in \mathrm{Cl}_{\mathcal{O}_{\mathcal{K}}}$ can be written as $[\mathfrak{a}] = [\mathfrak{b}_1]^{e_1} \cdots [\mathfrak{b}_k]^{e_k}$ for some $\mathfrak{b}_1, \ldots, \mathfrak{b}_k \in \mathcal{B}$ and integers $e_1, \ldots, e_k$. Let $N = \#\mathcal{B}$ and $\mathcal{B} = \{\mathfrak{b}_1, \ldots, \mathfrak{b}_N\}$. Then there is a surjective homomorphism φ from $\mathbb{Z}^N$ to $\mathrm{Cl}_{\mathcal{O}_{\mathcal{K}}}$ which takes $(e_1, \ldots, e_N)$ to $[\mathfrak{b}_1]^{e_1} \cdots [\mathfrak{b}_N]^{e_N}$. By the isomorphism theorem, $\mathrm{Cl}_{\mathcal{O}_{\mathcal{K}}}$ is isomorphic to $\mathbb{Z}^N/\mathrm{Ker}(\varphi)$. It is easy to see that $\mathrm{Ker}(\varphi)$ is a sub-lattice of $\mathbb{Z}^N$.

In view of the above, computing $\mathrm{Cl}_{\mathcal{O}_{\mathcal{K}}}$ boils down to determining $\mathcal{B}$ and computing $\mathrm{Ker}(\varphi)$. For the moment suppose that $\mathcal{B}$ is given. A relation is an equation of the form

$$\mathfrak{b}_1^{e_1} \cdots \mathfrak{b}_N^{e_N} \quad = \quad \langle x \rangle. \tag{3}$$

Suppose A is an $m \times N$ matrix whose rows are vectors $(e_1, \ldots, e_N) \in \mathbb{Z}^N$ corresponding to relations of the form given by (3). The structure of $\mathrm{Cl}_{\mathcal{O}_{\mathcal{K}}}$ is given by the Smith normal form (SNF) of A.

Obtaining relations of the form (3) forms the main computational task of the algorithm for computing the structure of the class group. The linear algebra part consists of computing the SNF of the relation matrix A. There is a further verification step that is required. The SNF of A possibly provides a factor of the class number. One computes an approximation of the regulator from the obtained relations and verifies that the product of the class number and the regulator is approximately one. If this does not hold, then the SNF of A provides a proper factor of the class number. In that case, more relations need to be obtained and the linear algebra and verification steps repeated. For more details of these two steps we refer to [7, 9].

The set $\mathcal{B}$ is defined to be the set of all prime ideals whose norms are at most a constant B. Then $\#\mathcal{B} \approx B/\log B$. Assuming the Extended Riemann Hypothesis, Bach [1] showed that choosing B equal to $12(\log|\Delta_{\mathcal{K}}|)^2$ is sufficient to ensure that the classes of the ideals in $\mathcal{B}$ generate the entire class group. By Bach's bound we mean the quantity $12(\log|\Delta_{\mathcal{K}}|)^2$ and denote it by C. Choosing $B = C$ is not sufficient to ensure that sufficiently many relations can be found. The value of B is chosen to be higher than Bach's bound. Following Gélin [9], B is set to be equal to $L_{|\Delta_{\mathcal{K}}|}(\delta, c_b)$. For appropriate choices of δ and c_b the overall runtime of the algorithm is sub-exponential. We refer to [9] for details.

A crucial issue in the relation collection has been pointed out by Gélin [9]. The relation collection phase is completed when the number of relations is larger than $\#\mathcal{B}$ and when all ideals of norms below Bach's bound are involved in at least one relation.

2.1 Generating Relations

Suppose as above that $\mathcal{B} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_N\}$ is the set of all prime ideals having norms less than B where B itself is at least as large as $12(\log|\Delta_{\mathcal{K}}|)^2$. For the asymptotic analysis, B is chosen to be $B = L_{|\Delta_{\mathcal{K}}|}(\delta, c_b)$ for suitable values of δ and c_b; this point is discussed later.

The set $\mathcal{B}$ is said to be the factor basis. For $x \in \mathcal{O}_{\mathcal{K}}$ and integers $e_1, \ldots, e_N$, a relation is given by

$$\mathfrak{p}_1^{e_1} \cdots \mathfrak{p}_N^{e_N} \quad = \quad \langle x \rangle. \tag{4}$$

The main task of the class group computation algorithm is to generate relations of the form (4).

The basic idea for obtaining relations is to generate random principal ideals and then check whether it is smooth over $\mathcal{B}$. In practice, given a principal ideal $\langle x \rangle$, one checks whether the positive integer $\mathcal{N}(\langle x \rangle) = |\mathcal{N}(x)|$ is B-smooth. If this holds, then $\langle x \rangle$ is factored.

Given a random element of x of $\mathcal{O}_\mathcal{K}$, the probability of $|\mathcal{N}(x)|$ being B-smooth is very small. So, simply trying out random elements of $\mathcal{O}_\mathcal{K}$ will not lead to an efficient algorithm. To ensure that the probability of $|\mathcal{N}(x)|$ being B-smooth is reasonable, it is required to choose x in a manner which ensures that $|\mathcal{N}(x)|$ satisfies some pre-determined upper bound. The literature contains various methods of choosing x such that $|\mathcal{N}(x)|$ is below a desired bound.

Given an element $\alpha \in \mathcal{O}_\mathcal{K}$, its canonical embedding $\sigma(\alpha)$ into $\mathbb{C}^n$ is $\sigma(\alpha) = (\sigma_1(\alpha), \ldots, \sigma_n(\alpha))$. Since the complex embeddings occur in pairs, the vector $(\sigma_1(\alpha), \ldots, \sigma_n(\alpha))$ can be considered to be represented by a vector from $\mathbb{R}^n$. So, henceforth, we will consider $\sigma(\alpha)$ be a vector in $\mathbb{R}^n$. Consider the lattice $\sigma(\mathfrak{a})$ obtained by the embedding of an ideal $\mathfrak{a}$ of $\mathcal{O}_\mathcal{K}$. Suppose v is a short vector in $\sigma(\mathfrak{a})$ and let x_v be the corresponding element in $\mathfrak{a}$ such that $\sigma(x_v) = v$. Then $|\mathcal{N}(x_v)|$ is small and so the principal ideal $\langle x_v \rangle$ also has small norm.

Algorithm 1 describes Gélin's method of generating relation using ideal reduction. The parameters of the algorithm are k, A, β and B. The parameter k determines the number of ideals to be considered in each iteration, A determines the maximum value of the exponent to which the ideals are raised, the parameter β determines the block size of the BKZ-reduction, and the parameter B is the bound on the norms of the ideals in the factor basis. Gélin specifies k and A to be poly$(\log |\Delta_\mathcal{K}|)$. Further, the value of β is to be chosen such that the overall complexity is subexponential in $|\Delta_\mathcal{K}|$.

In Step 5 of Algorithm 1, a single vector from the BKZ reduction is returned. It is, however, possible that several vectors from the BKZ reduction have sufficiently small norms and a linear combination of these vectors can be tried. This saves the number of BKZ reductions and leads to practical speed-ups. Such improvement does not change the asymptotic complexity. We refer to [9] for details of this improvement and other implementation notes.

Algorithm 1: Gélin's method for generating relation using ideal reduction [9].

```
   Input: The factor base $\mathcal{B} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_N\}$.
   Output: The set of generated relations.
1  while sufficient relations have not been obtained do
2      Choose $k$ random prime ideals $\mathfrak{p}_{j_1}, \mathfrak{p}_{j_2}, \ldots, \mathfrak{p}_{j_k}$ from $\mathcal{B}$
3      Choose $k$ random exponents $e_{j_1}, e_{j_2}, \ldots, e_{j_k}$ from $\{1, 2, \ldots, A\}$
4      Set $\mathfrak{a} = \prod_{i=1}^{N} {\mathfrak{p}_i}^{e_i}$ with $e_i = 0$ if $i \notin \{j_1, \ldots, j_k\}$
5      Compute the smallest element $v$ of the BKZ$_\beta$ reduced basis of the lattice $\sigma(\mathfrak{a})$
6      Obtain the algebraic integer $x_v$ corresponding to $v$
7      Set $\mathfrak{b}$ as the unique ideal such that $\langle x_v \rangle = \mathfrak{a}\mathfrak{b}$
8      if $|\mathcal{N}(\mathfrak{b})|$ is $B$-smooth then
9          Obtain the factorization of $\mathfrak{b}$ such that $\mathfrak{b} = \prod_{i=1}^{N} {\mathfrak{p}_i}^{e'_i}$
10         Store the relation $\langle x_v \rangle = \prod {\mathfrak{p}_i}^{e_i + e'_i}$
```

Step 8 of Algorithm 1 checks whether $\mathcal{N}(\mathfrak{b})$ is B-smooth. The actual requirement is that the ideal $\mathfrak{b}$ is $\mathcal{B}$-smooth. If $\mathcal{N}(\mathfrak{b})$ is B-smooth, then it follows that $\mathfrak{b}$ is $\mathcal{B}$-smooth and a factorisation of $\mathfrak{b}$ can be obtained from a factorisation of $\mathcal{N}(\mathfrak{b})$. We refer to [9] for details.

Notation: Before proceeding further, we fix some notation.

B	upper bound on the size of the ideals in the factor basis;
$\mathcal{B}$	the factor basis;
N	number of prime ideals in the factor basis, i.e., $N = \#\mathcal{B}$;
C	Bach's bound, i.e., $C = 12(\log |\Delta_\mathcal{K}|)^2$;
$\mathcal{C}$	the set of prime ideals whose norms are at most C;
N_b	number of prime ideals whose norms are at most C, i.e., $N_b = \#\mathcal{C}$;
β	block size of BKZ reduction.

3 Generating Relations from a Pseudo-Random Walk on Ideals

Consider Algorithm 1. The asymptotic complexity of the algorithm is determined by B and β. The parameter β determines the upper bound on the norm of the element x_v; B and β determine the smoothness probability of the ideal $\langle x_v \rangle$.

We do not propose to change these parameters. Instead, we focus on the concrete complexity of an individual iteration. This consists of computing the ideal $\mathfrak{a}$, computing the BKZ-reduction, and the smoothness check of the norm of $\langle x_v \rangle$. The portions on BKZ-reduction and the smoothness check are also not modified. Our focus is on reducing the cost of computing the ideal $\mathfrak{a}$.

In Step 4 of Algorithm 1, the ideal $\langle \mathfrak{a} \rangle$ is computed as

$$\mathfrak{a} \quad = \quad \prod_{i=1}^{k} \mathfrak{p}_{j_i}^{e_{j_i}}. \tag{5}$$

Since each e_{j_i} is chosen randomly from $\{1, 2, \ldots, A\}$, computing $\mathfrak{p}_{j_i}^{e_{j_i}}$ requires about $\log_2 A$ ideal multiplications. So, the total cost of computing $\mathfrak{a}$ in (5) is about $(k-1)\log_2 A$ ideal multiplications. Below we describe a new algorithm for generating $\langle \mathfrak{a} \rangle$ which requires a single ideal multiplication.

Our idea is to perform a pseudo-random walk on a sufficiently large set of ideals. Each step of the walk will generate an ideal $\mathfrak{a}$ to which the the BKZ-reduction and the rest of Algorithm 1 can be applied. The cost of each step will consist of a single ideal multiplication. The pseudo-random walk that we construct is inspired by Pollard's rho algorithm.

Suppose $[\mathfrak{t}_1, \ldots, \mathfrak{t}_m]$ is a list of m pre-computed ideals. Let $\mathcal{H}$ be a hash function which maps an ideal to the set $\{0, \ldots, m-1\}$. For $i \geq 0$, the pseudo-random walk proceeds as follows. An ideal $\mathfrak{a}_0$ is chosen and for $i \geq 1$, a_i is obtained as follows: let $m_i = \mathcal{H}(\mathfrak{a}_{i-1})$ and set $\mathfrak{a}_i = \mathfrak{a}_{i-1} \cdot \mathfrak{t}_{m_i}$.

The list $[\mathfrak{t}_1, \ldots, \mathfrak{t}_m]$ and some additional information are stored in a table $\mathfrak{T}$. We explain how the ideals $\mathfrak{t}_1, \ldots, \mathfrak{t}_m$ are constructed and the entries of the table $\mathfrak{T}$.

Let $\mathcal{C} = \{\mathfrak{q}_1, \ldots, \mathfrak{q}_{N_b}\}$ be the prime ideals whose norms are below Bach's bound. Let κ be a parameter and set $q = \lfloor N_b/\kappa \rfloor$ and $r = N_b - q\kappa$ so that we have $N_b = q\kappa + r = r(\kappa+1) + (q-r)\kappa$. The set $\mathcal{C}$ is randomly partitioned into q groups where the first r groups each have $\kappa+1$ ideals and the last $q-r$ groups each have κ ideals. The ideals in each of the groups are multiplied together and the products are stored in $\mathfrak{T}$. Along with the product ideal, the information identifying the ideals that have been multiplied to obtain the product is also stored in $\mathfrak{T}$. The random paritioning of the ideals in $\mathcal{C}$ into groups, multiplying together the ideals in each group and storing them in table $\mathfrak{T}$ is carried out a total of R times. So, the number of entries in $\mathfrak{T}$ is qR, i.e., $m = qR$. Further, each ideal in $\mathcal{C}$ is represented a total of R times in the table $\mathfrak{T}$. The method for constructing $\mathfrak{T}$ is shown in Algorithm 2.

There are a total of Rq entries in $\mathfrak{T}$, i.e., $m = Rq$. The entries of $\mathfrak{T}$ are pairs. For $0 \leq i \leq m-1$, $\mathfrak{T}[i]$ is an entry of the form $(\mathfrak{b}, (j_1, \ldots, j_s))$. By $\mathfrak{T}[i].\mathsf{ideal}$ we will denote the ideal $\mathfrak{b}$ and by $\mathfrak{T}[i].\mathsf{index}$ we will denote the tuple $(j_1, \ldots, j_s)$.

The table $\mathfrak{T}$ is constructed in a pre-computation phase. This pre-computation consists of about $Rq\kappa \approx RN_b$ ideal multplications. When R is a constant, the number of ideal multiplications required to construct $\mathfrak{T}$ is negligible with respect to the number of ideal multiplications required in all the iterations for relation collection.

How long should the pseudo-random walk proceed? There are several aspects to this question.

1. As the walk progresses, both the number and the exponents of the prime ideals occuring as factors in the ideals visited by the walk increases. So, a long walk can result in ideals with large norms.

2. From a practical point of view, in our Magma implementation we have observed that as the norms of the ideals increase, it becomes difficult to construct the lattices corresponding to the ideals.

In view of the above two points, long walks are not feasible, at least for Magma implementation. One way is to continue the walk as long as it is feasible to construct the associated lattice. Alternatively, one may

Algorithm 2: Construction of the pre-computed table $\mathfrak{T}$.

```
Input: The set of prime ideals $\mathfrak{C} = \{\mathfrak{q}_1, \ldots, \mathfrak{q}_{N_b}\}$ whose norms are below Bach's bound.
Output: The table $\mathfrak{T}$.
$q \leftarrow \lfloor N_b/\kappa \rfloor$, $r \leftarrow N_b - q\kappa$
$\mathfrak{T} \leftarrow \emptyset$
$\mathcal{J} \leftarrow \{1, 2, \ldots, N_b\}$
for $i_1 \leftarrow 1$ to $R$ do
    $\mathcal{I} \leftarrow \mathcal{J}$
    for $i_2 \leftarrow 1$ to $q$ do
        if $i_2 \leq r$ then
            $s \leftarrow (\kappa + 1)$
        else
            $s \leftarrow \kappa$
        $\{j_1, j_2, \ldots, j_s\} \leftarrow$ random set of $s$ distinct integers chosen from $\mathcal{I}$
        $\mathcal{I} \leftarrow \mathcal{I} \setminus \{j_1, j_2, \ldots, j_s\}$
        $\mathfrak{b} \leftarrow \mathfrak{q}_{j_1} \cdots \mathfrak{q}_{j_s}$
        Append $(\mathfrak{b}, (j_1, \ldots, j_s))$ to $\mathfrak{T}$
```

put an *a priori* bound on the length of an individual walk. Determining the bound requires performing some initial experiments to obtain an idea of the number of steps that the walk can proceed without encountering the failure of lattice construction.

The algorithm for relation collection based on the pseudo-random walk is shown in Algorithm 3. The parameters β and B are the same as those in Algorithm 1. It is assumed that the table $\mathfrak{T}$ has been constructed prior to the execution of Algorithm 3. Recall that the ideals stored in $\mathfrak{T}$ are products of either κ or $\kappa + 1$ prime ideals whose norms are below N_b. Algorithm 3 uses an additional parameter κ_0 which determines the number of prime ideals to be multiplied together to obtain the starting ideal of a walk. The variable wlen records the current length of the walk, while the parameter λ represents the maximum length of each walk.

As above, let $\mathcal{C} = \{\mathfrak{q}_1, \ldots, \mathfrak{q}_{N_b}\}$ be the set of all prime ideals whose norms are below Bach's bound. The actual factor basis $\mathcal{B} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_N\}$ consists of all prime ideals whose norms are below the bound B. We assume that for $i = 1, \ldots, N_b$, $\mathfrak{p}_i = \mathfrak{q}_i$, i.e., the first N_b prime ideals in $\mathcal{B}$ have norms below Bach's bound.

In Algorithm 3, the ideals $\mathfrak{a}$ which are visited by the pseudo-random walk are products of prime ideals whose norms are below C. This is because a walk starts with such an ideal and for each step, the present ideal is multiplied with an ideal from the pre-computed table. Since the ideals in the pre-computed table are also of the same type, the property of being products of prime ideals whose norms are below C holds for all the ideals $\mathfrak{a}$ visited by the walk. On the other hand, the smoothness check of $\mathcal{N}(\mathfrak{b})$ is with respect to B. As a result, the algorithm tries to ensure that $\mathfrak{b}$ is smooth over the factor basis $\mathcal{B}$ rather than $\mathcal{C}$. Trying to ensure the smoothness of $\mathfrak{b}$ over $\mathcal{C}$ will result in the theoretical smoothness probability being too low. In practice, however, it may be possible to work with $\mathcal{C}$ as the factor basis as we discuss later. Recall that one of the stopping criterion for relation collection is that each element of $\mathcal{C}$ is involved in at least one relation. Since the ideals $\mathfrak{a}$ are products of ideals from $\mathcal{C}$, this criterion becomes somewhat easier to ensure.

Remarks:

1. The list exp in Algorithm 3 is an array of N integers. This list is likely to be very sparse. So, it would be more efficient to represent exp as a list of pairs $[(i_1, e_1), \ldots, (i_s, e_s)]$, such that $\texttt{exp}[i] = e_j$ if $i = i_j$, $j = 1, \ldots, s$ and $\texttt{exp}[i] = 0$, otherwise. Another possibility is to represent exp as a list

Algorithm 3: Relation collection using pseudo-random walk.

```
Input: The factor base $\mathcal{B} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_N\}$.
Output: The set of generated relations.
1  while Sufficient relations have not been found do
2      $\mathsf{exp}[i] = 0$ for $i = 1, \ldots, N$
3      Choose $s$ randomly from $\{1, \ldots, \kappa_0\}$
4      Choose $i_1, \ldots, i_s$ randomly from $\{1, \ldots, N_b\}$
5      Set $\mathfrak{a} = \mathfrak{p}_{i_1} \cdots \mathfrak{p}_{i_s}$
6      $\mathsf{exp}[i] \leftarrow 1$ for $i = i_1, \ldots, i_s$;
7      $\mathsf{wlen} \leftarrow 1$;
8      while $\mathsf{wlen} \leq \lambda$ do
9          Compute the smallest element $v$ of the BKZ$_\beta$ reduced basis of the lattice $\sigma(\mathfrak{a})$
10         Obtain the algebraic integer $x_v$ corresponding to $v$
11         if $|\mathcal{N}(x_v)/\mathcal{N}(\mathfrak{a})|$ is $B$-smooth then
12             Set $\mathfrak{b}$ as the unique ideal such that $\langle x_v \rangle = \mathfrak{a}\mathfrak{b}$
13             Obtain the factorization of $\mathfrak{b}$ such that $\mathfrak{b} = \prod_{i=1}^{N} p_i{}^{e'_i}$
14             for $i \leftarrow 1$ to $N$ do
15                 $\mathsf{exp}[i] \leftarrow \mathsf{exp}[i] + e'_i$
16             Store the relation $\langle x_v \rangle = \prod_{i=1}^{N} \mathfrak{p}_i{}^{\mathsf{exp}[i]}$
17         $\ell \leftarrow \mathcal{H}(\mathfrak{a})$
18         $\mathfrak{a} \leftarrow \mathfrak{a} \cdot \mathfrak{T}[\ell].\mathsf{ideal}$
19         $\mathsf{exp} \leftarrow \mathsf{exp} + \mathfrak{T}[\ell].\mathsf{index}$
20         $\mathsf{wlen} \leftarrow \mathsf{wlen} + 1$;
```

of integers where i_1 is repeated e_1 times, i_2 is repeated e_2 times, and so on. Since the integers $e_1, e_2, \ldots$ are quite likely to be equal to 1, this representation would be even more compact than storing the pairs $(i_1, e_1), (i_2, e_2), \ldots$. The operations on exp are to be suitably modified to be used with a compact representation.

2. The computation of the ideal $\mathfrak{b}$ in Step 7 of Algorithm 1 requires computing $\mathfrak{a}^{-1}$. Since $\mathcal{N}(\mathfrak{b})$ may not turn out to be B-smooth in Step 8, the computation of the ideal $\mathfrak{b}$ may actually not be required. Slightly altering the sequence of instructions, it is possible to avoid the computation of $\mathfrak{a}^{-1}$ in the cases where $\mathcal{N}(\mathfrak{b})$ is not B-smooth. Since $\mathcal{N}(\mathfrak{b}) = \mathcal{N}(x_v)/\mathcal{N}(\mathfrak{a})$, Algorithm 3 checks for the B-smoothness of $|\mathcal{N}(x_v)/\mathcal{N}(\mathfrak{a})|$ and computes $\mathfrak{b}$ only if the check is successful.

3. Note that the norms of the ideals in $\mathfrak{T}$ are known. The table $\mathfrak{T}$ can be expanded to store the norms of the ideals. This speeds up the computation of the norm of $\mathfrak{a}$ required in Step 11 of Algorithm 3. Along with the current ideal $\mathfrak{a}$ of the walk, its norm is also stored. The norm of the next ideal in the walk is obtained by multiplying the norm of the current ideal $\mathfrak{a}$ and the norm of the ideal in the location $\mathfrak{T}[\mathcal{H}(\mathfrak{a})]$.

4. The pseudo-random walk is a sequential procedure. Parallelism can be incorporated by starting independent pseudo-random walks. The table $\mathfrak{T}$ remains the same for all the walks. The starting ideals are chosen independently. This allows the separate walks to proceed independently.

5. Algorithm 3 has been described keeping in mind that the precision is fixed. Ideals in the initial steps of the walk have smaller norms compared to ideals in the later stages of the walk. So, an alternative approach would be to start with a smaller precision and increase the precision as the walk progresses. Since precision determines the efficiency of the various computational steps, working with a lower precision in the initial stages would lead to improved efficiency. The idea of increasing precision as

the walk progresses can work up to a certain point, since as the precision becomes too large, the computation slows down considerably. At that point, it is better to switch to a new initial point and start a new walk.

We compare the number of ideal multiplications required by Algorithms 1 and 3. For the comparison, we ignore the number of ideal multiplications required to prepare the table $\mathfrak{T}$. This is a one-time activity whose cost has been mentioned above; amortised over the iterations required for generating all the relations, this cost is negligible. We revisit this issue with respect to the experiments that have been conducted. A pseudo-random walk in Algorithm 3 proceeds for λ steps. These λ steps require at most $\kappa_0 + \lambda - 2$ ideal multiplications. In comparison, Algorithm 1 requires about $\lambda(k-1)\log_2 A$ ideal multiplications for generating λ ideals $\mathfrak{a}$ in λ iterations. The other costs of both Algorithms 1 and 3, namely BKZ-reduction, smoothness checking and ideal factorisation, remain the same, though there is the issue of the norms of the ideals $\mathfrak{a}$ to be considered. We discuss this point below.

Though in principle, the reduction in the number of ideal multiplications should lead to improvement in time, there is a practical aspect that needs to be kept in mind. In practice, the time to multiply two ideals depends on the norms of the ideals. Even if one of the ideals has a relatively large norm, there is a noticeable increase in the time to compute the product. While determining the walk length, this aspect needs to be kept in the mind. As the walk length increases, so does the norms of the ideals visited by the walk. This means that even though each step of the walk requires a single multiplication, the time for this multiplication increases with the length of the walk. So, if a walk is too long, then it may turn out that computing the next ideal of the walk takes very long time. In effect, this means that for Algorithm 3 to be competitive with Algorithm 1, the walk length should not be too long. In our experiments, we have fixed the walk length so that the norms of the ideals visited by the walk in Algorithm 3 are less than the norms of the ideals generated by Algorithm 1. In later sections, we provide experimental results from our Magma implementation to show that such a strategy indeed provides a substantial improvement in the relation collection time.

We further consider the issue of increase in the norms of the ideals visited by the walk. The first ideal in the walk has norm at most $\kappa_0 C$. At each step, the ideal in the walk is multiplied by κ or $\kappa+1$ prime ideals having norms at most C. So, the ideal obtained after i steps of the walk has norm at most $(\kappa_0 + i(\kappa+1))C$ which is at most $(\kappa_0 + \lambda(\kappa+1))C$ since $i \leq \lambda$. In comparison, the ideals $\mathfrak{a}$ generated in each iteration of Algorithm 1 have norms to be at most about $kAB/2$. So, the norms of the ideals in the walk in Algorithm 3 are at most the norms of ideals the ideals $\mathfrak{a}$ in Algorithm 1 if the condition $(\kappa_0 + \lambda(\kappa+1))C \leq kAB/2$ holds. The parameters may be chosen to satisfy this condition. Note that in Algorithm 1, the norms of all the ideals are about $kAB/2$, while for Algorithm 3, the norms of the ideals in the initial steps of the walk are lower.

A pseudo-random walk chooses the first ideal randomly while the subsequent ideals are chosen deterministically. A crucial requirement in the asymptotic analysis of the algorithm is the smoothness probability of random ideals. Since the ideals appearing in a walk do not have independent randomness, one has to heuristically assume that the result on the smoothness probability of random ideals applies to the ideals occurring in the walk. Our experiments show that there is no substantial effect on the smoothness probability.

Let Π denote the probability that an ideal $\mathfrak{a}$ in the walk leads to an ideal $\mathfrak{b}$ computed in Step 12 of Algorithm 3 which is smooth over $\mathcal{B}$ and hence leads to a relation. About $1/\Pi$ ideals need to be considered to obtain a single relation. A total of about N relations are required. Consequently, about N/Π ideals need to be considered to obtain all the relations. Each walk visits λ ideals. The total number of walks required to consider N/Π ideals is $N/(\Pi\lambda)$. The total number of starting points is $\sum_{s=1}^{\kappa_0}\binom{N_b}{s}$. So, to ensure that $N/(\Pi\lambda)$ walks are possible, we must have

$$\sum_{s=1}^{\kappa_0}\binom{N_b}{\kappa_0} \geq \frac{N}{\Pi\lambda}. \tag{6}$$

The parameter κ_0 has to be chosen to satisfy (6).

As proved in [9], the norm of $\mathfrak{b}$ considered in Step 12 of Algorithm 3 satisfies the bound $\mathcal{N}(\mathfrak{b}) \leq \beta^{n(n-1)/(2(\beta-1))}\sqrt{|\Delta_{\mathcal{K}}|}$. Further, the B-smoothness of $\mathcal{N}(\mathfrak{b})$ depends on the value of B. We do not suggest any change to either the value of β or to the value of B. The difference between Algorithms 1 and 3 is in the generation of the ideal $\mathfrak{a}$. As discussed above, the norm of $\mathfrak{a}$ considered by Algorithm 3 is never more than the norm of $\mathfrak{a}$ considered by Algorithm 1. The net effect of all these considerations is that the asymptotic result obtained in [9] for Algorithm 1 also holds for Algorithm 3. The advantage of Algorithm 3 over Algorithm 1 is in improved practical efficiency.

4 Implementation

We have implemented Algorithms 1 and 3 using Magma, version V2.22-3. We did not perform the entire class group computation. Rather, we performed two kinds of experiments. The set of experiments compares the times required for relation collection by Algorithms 1 and 3. These experiments show that in general Algorithm 3 performs better than Algorithm 1. The second set of experiments performs the entire relation collection step for two fields having discriminants of sizes about 157 and 256 bits. This demonstrates that Algorithm 3 can actually work in practice.

Various issues arise while implementing Algorithm 1. For guidance in determining parameters, we considered the asymptotic analysis. This, however, fails to provide sufficient information and in some cases lead to substantially less efficient parameter choices. Below, we mention these issues as they arise in the description of our implementation effort.

4.1 Choice of Number Fields

Gélin and Joux [12] provide the following classification of number fields. Let $n_0 > 1$ be a real parameter arbitrarliy close to 1, $d_0 > 0$, $\alpha \in [0,1]$ and $\gamma \geq (1-\alpha)$. The class $\mathcal{D}_{n_0,d_0,\alpha,\gamma}$ is defined to be the set of all number fields $\mathcal{K}$ with discriminant $\Delta_{\mathcal{K}}$ having a monic defining polynomial $T \in \mathbb{Z}[X]$ of degree n such the following inequalities hold true.

$$\frac{1}{n_0}\left(\frac{\log(|\Delta_{\mathcal{K}}|)}{\log(\log(|\Delta_{\mathcal{K}|}))}\right)^{\alpha} \leq n \leq n_0\left(\frac{\log(|\Delta_{\mathcal{K}}|)}{\log(\log(|\Delta_{\mathcal{K}}|))}\right)^{\alpha} \tag{7}$$

and

$$\log(\|T\|_{\infty}) \quad \leq \quad d_0(\log(|\Delta_{\mathcal{K}}|)^{\gamma}\log(\log(|\Delta_{\mathcal{K}}|))^{(1-\gamma)}. \tag{8}$$

Here, $\|T\|_{\infty}$ denotes the maximum of the absolute values of the coefficients of T. This parameter has also been called the height of the polynomial T. It has been shown in Gélin [9] that asymptotically the classification covers all number fields.

The actual number fields that we have used have been chosen from the online database of number fields [14]. We considered four number fields defined by the following four polynomials.

$$\begin{aligned}
T_1(X) &= X^{10} - 20X^8 - 170X^6 - 1704X^5 - 2100X^4 - 1680X^3 - 23865X^2 - 36360X + 15984,\\
T_2(X) &= X^{15} - 15X^{13} + 105X^{11} - 78X^{10} - 425X^9 + 780X^8 + 1050X^7 - 3510X^6 - 2832X^5\\
&\quad +7800X^4 + 7660X^3 - 7800X^2 - 13320X - 8856,\\
T_3(X) &= X^{20} - 5X^{19} + 76X^{18} - 247X^{17} + 1197X^{16} - 8474X^{15} + 15561X^{14} - 112347X^{13} + 325793X^{12}\\
&\quad -787322X^{11} + 3851661X^{10} - 5756183X^9 + 20865344X^8 - 48001353X^7 + 45895165X^6\\
&\quad -245996344X^5 + 8889264X^4 - 588303992X^3 - 54940704X^2 - 538817408X + 31141888,\\
T_4(X) &= X^{25} - 344X^{23} - 316X^{22} + 45906X^{21} + 78964X^{20} - 3003991X^{19} - 7163070X^{18} + 101409224X^{17}\\
&\quad +293673294X^{16} - 1740399699X^{15} - 5640650024X^{14} + 15351660849X^{13} + 53959254132X^{12}\\
&\quad -67237888386X^{11} - 259371867838X^{10} + 117450950109X^9 + 587040491084X^8 + 30969137155X^7 - 547923508138X^6\\
&\quad -206267098153X^5 + 109439981776X^4 + 40995170780X^3 - 9046378504X^2 - 1197994128X + 80434784.
\end{aligned}$$

field	poly	n_0	α	$\log_2\lvert\Delta\rvert$
$\mathcal{K}_1$	$T_1(X)$	1.10	0.935	63.5
$\mathcal{K}_2$	$T_2(X)$	1.11	0.990	81.5
$\mathcal{K}_3$	$T_2(X)$	1.01	0.952	157.2
$\mathcal{K}_4$	$T_2(X)$	1.01	0.911	256.2

Table 1: The number fields used in this work along with the values of n_0 and α used in the classification from [9]. Also, the size of the discriminant is shown for each field.

α	β	c_b	B	complexity of reln coll	overall complexity
$[1/2, 3/4]$	$(\log\lvert\Delta_{\mathcal{K}}\rvert)^{1/2}$	$\frac{1}{2\sqrt{\omega}}$	$L_{\lvert\Delta_{\mathcal{K}}\rvert}\left(\frac{1}{2}, c_b\right)$	$L_{\lvert\Delta_{\mathcal{K}}\rvert}\left(\frac{1}{2}, \frac{1}{4c_b} + c_b\right)$	$L_{\lvert\Delta_{\mathcal{K}}\rvert}\left(\frac{1}{2}, \frac{\omega+1}{2\sqrt{\omega}}\right)$
$(3/4, 1]$	$(\log\lvert\Delta_{\mathcal{K}}\rvert)^{2\alpha/3}$	$\sqrt{\frac{2\alpha c}{3}}$	$L_{\lvert\Delta_{\mathcal{K}}\rvert}\left(\frac{2\alpha}{3}, c_b\right)$	$L_{\lvert\Delta_{\mathcal{K}}\rvert}\left(\frac{2\alpha}{3}, 2c_b\right)$	$L_{\lvert\Delta_{\mathcal{K}}\rvert}\left(\frac{2\alpha}{3}, o(1)\right)$

Table 2: Choices of β and B and the corresponding asymptotic complexities provided in [9]. In the table, ω is the exponent of linear algebra and determination of c is explained in Section 5.2 of [9].

Given a number field, the values of n_0, d_0, α and γ required in the Gélin-Joux classification are not unique. The definition mentions that $n_0 > 1$ and also that n_0 is arbitrarily close to 1. For a concrete number field, interpreting the latter condition is difficult. In fact, the values of n_0 and α need to be simultaneously determined so that (7) holds. Another important issue is that for $\alpha \in (3/4, 1]$, the value of α determines the complexity of the algorithm as can be seen from the second row of Table 2. The lower the value of α, the lower the complexity. So, there is a motivation to choose n_0 and α such that α is as small as possible. It may be noted from (7) that if we assume that n_0 equals 1, then the value of α is equal to $\log n/\nu$, with $\nu = \log\lvert\Delta_{\mathcal{K}}\rvert/\log(\log\lvert\Delta_{\mathcal{K}}\rvert)$. The definition, however, requires $n_0 > 1$; further, the value of $\log n/\nu$ can turn out to be greater than 1 as is the case for the field $\mathcal{K}_3$ shown in Table 1. So, to determine the values of α and n_0, one may use the value $\log n/\nu$ as a starting guideline and then try to reduce the value of α as much as possible. We have followed this strategy. Additionally, we tried to determine n_0 and α such that the only integer in the range determined by the bounds in (7) is n, since this implies that the pair (α, n_0) uniquely determines n. It was possible to achieve this condition for the values of $n = 10, 20$ and 25; for $n = 15$, however, the range includes the integers 13, 14 and 15 and we found it difficult to tune the values of n_0 and α such that only 15 is in the desired range. The values of n_0, α and the size of the discriminant for the four number fields are shown in Table 2.

For the classification of the number fields, the values of d_0 and γ are also required to be determined. The conditions on these two parameters are that $d_0 > 0$ and $\gamma \geq 1 - \alpha$. From Table 2, the complexity of the algorithm does not depend on the values of d_0 and γ. So, one simple method of determining d_0 and γ is to fix the value of γ to be equal to $1 - \alpha$ and choose d_0 to be the least integer satisfying from (8). There are other strategies such as setting $d_0 = 1$ and then determining γ from (8). This strategy, however, does not necessarily ensure that $\gamma \geq 1 - \alpha$, though for the chosen fields, this condition holds. Since the values of d_0 and γ do not affect the complexity of the algorithm, we do not provide the values of these parameters.

4.2 Determining B and β

For a number field $\mathcal{K}$, based on the value of α, Gélin [9] provides the appropriate choices of β and B and the corresponding complexities. These are shown in Table 2. Note that $\alpha \geq 1/2$ in Table 2. In [9], number fields for which $\alpha \geq 1/2$ have been termed large degree number fields.

To implement Algorithm 1 (and also Algorithm 3), it is necessary to fix the values of β and B. For a given number field $\mathcal{K}$, the value of $\lvert\Delta_{\mathcal{K}}\rvert$ is known. Using this value of $\lvert\Delta_{\mathcal{K}}\rvert$ in the expressions for β and B given in Table 2, it is possible to find concrete values of β and B. For the number fields that we have

field	β_{th}	B	C
$\mathcal{K}_1$	10	$970357 \approx 2^{19.9}$	$23222 \approx 2^{14.5}$
$\mathcal{K}_2$	15	$132526880 \approx 2^{27.0}$	$38262 \approx 2^{15.2}$
$\mathcal{K}_3$	20	$883661511529 \approx 2^{39.7}$	$142426 \approx 2^{17.1}$
$\mathcal{K}_4$	24	$997850477380847 \approx 2^{49.8}$	$378313 \approx 2^{18.5}$

Table 3: Values of β_{th}, B and Bach's bound C for the number fields in Table 1. For the computation of B, the value of c in Table 2 has been assumed to be 1.

used, the value of α lies in the range $(3/4, 1]$.

We would like to highlight that asymptotically speaking B is much larger than C. The value of C is $12(\log|\Delta_{\mathcal{K}}|)^2$ which is $L_{|\Delta_{\mathcal{K}}|}(0, \cdot)$, whereas B is $L_{|\Delta_{\mathcal{K}}|}\left(\frac{2\alpha}{3}, c_b\right)$. Below we discuss the several issues related to using the asymptotic expression in deriving an appropriate value of B for concrete number fields. To do this, we need to evaluate the L-notation for concrete values of $|\Delta_{\mathcal{K}}|$ and the arguments. From (1), we note that there is an $o(1)$ term in the definition of the sub-exponential notation. We take this term to be 0.

Determining the value of B for $\alpha \in (3/4, 1]$ requires determining the value of c. The asymptotic analysis in [9] mentions that c can be taken to be any positive value which is very close to 0. In the concrete setting, however, a low value of c (such as 10^{-3}) leads to the size of the factor basis being about one or two for all the fields in Table 1, which is useless. Since the asymptotic analysis does not help in determining the value of c for a concrete setting, somewhat arbitrarily, one may set the value of c to be 1 for determining the value of B. The corresponding values of B are shown in Table 3. In this context, we note that the value of c can be chosen so that the value of B becomes close to the value of C.

- For $\mathcal{K}_1$, choosing $c = 0.53$ leads to B being 22921, while C is 23222.
- For $\mathcal{K}_2$, choosing $c = 0.32$ leads to B being 39459, while C is 38262.
- For $\mathcal{K}_3$, choosing $c = 0.18$ leads to B being 117218, while C is 142426.
- For $\mathcal{K}_4$, choosing $c = 0.138$ leads to B being 373691, while C is 378313.

The value of β obtained from Table 2 is denoted by β_{th}. It turns out that the value of β_{th} is equal to n for $n = 15$ and 20; is equal to $n - 1$ for $n = 25$; and equal to $n + 1$ for $n = 10$. Since β_{th} is the block size, its value is at most n, so for $n = 10$, the value of β_{th} should be 10. The values of β_{th} indicate that for the fields under consideration, the BKZ$_{\beta_{th}}$ reduction returns a smallest vector in the lattice. The values of β_{th}, B as well as the values of C are shown in Table 3 for the various number fields in Table 1.

The values of B and β_{th} shown in Table 3 are obtained from the asymptotic analysis. There are significant difficulties in implementing Algorithm 1 using these values of B and β_{th}.

1. For running BKZ$_\beta$, Magma has a limitation. Our experiments show that while Magma is able to execute BKZ$_\beta$ for $\beta = 2$, for higher values of β this is not ensured. In several runs, we have seen that the Magma is sometimes able to perform the computation and sometimes the process terminates abnormally. Further, in the cases that Magma did complete the execution, the norms of the smallest elements returned by BKZ$_\beta$ for various values of $\beta > 2$ turned out to be similar in size of the norms of the smallest elements returned by BKZ$_2$. In view of this, we decided to fix $\beta = 2$ for the actual relation collection.

2. The construction of the ideal $\mathfrak{a}$ in Algorithm 1 requires randomly choosing k prime ideals from the factor basis. To be able to make such choices, it is required to generate and store the entire factor basis before Algorithm 1 can be run. For $\mathcal{K}_2$, $\mathcal{K}_3$ and $\mathcal{K}_4$, the size of the factor basis is quite large (for the value of B obtained by setting $c = 1$). Magma is unable to construct such a large factor basis. In particular, the factor bases for $\mathcal{K}_3$ and $\mathcal{K}_4$ are too large to be stored on our systems. We next discuss how we have handled this difficulty.

field	$(\mathfrak{M},\, \mathcal{P}(\mathfrak{M}, C))$	$(\beta,\, \mathfrak{B}_\beta,\, \mathcal{P}(\mathfrak{B}_\beta, B))$	N_b
$\mathcal{K}_1$	$(2^{16}, 0.897)$	$(2, 2^{76.75}, 2^{-7.51})$ $(10, 2^{48.36}, 0.116)$	2630
$\mathcal{K}_2$	$(2^{15}, \approx 1)$	$(2, 2^{145.75}, 2^{-13.13})$ $(15, 2^{70.05}, 0.084)$	4150
$\mathcal{K}_3$	$(2^{42}, 0.110)$	$(2, 2^{268.60}, 2^{-18.66})$ $(20, 2^{121.82}, 0.032)$	13097
$\mathcal{K}_4$	$(2^{74}, 0.004)$	$(2, 2^{428.10}, 2^{-26.68})$ $(24, 2^{187.90}, 0.007)$	32385

Table 4: Various smoothness probabilities and the number of ideals with norms at most C for the fields considered in this work.

As mentioned earlier, it has been proved in [9] that the norm of the ideal $\mathfrak{b}$ in Algorithm 1 satisfies the bound $\mathcal{N}(\mathfrak{b}) \leq \mathfrak{B}_\beta = \beta^{n(n-1)/(2(\beta-1))}\sqrt{|\Delta_\mathcal{K}|}$. A standard heuristic is that the probability $\mathcal{P}(x, y)$ that an ideal of norm bounded by x is y-smooth satisfies $\mathcal{P}(x, y) \geq \exp(-u(\log u)(1 + o(1)))$, where $u = \log x / \log y$. In [9], this heuristic is used in the asymptotic analysis of Algorithm 1 where x is taken to be equal to $\mathfrak{B}_\beta$ and y is taken to be equal to B. The expression $\mathfrak{B}_\beta$ is an *upper bound* on the norm of $\mathfrak{b}$. We ran some experiments to check the tightness of this bound. The experiments consisted of running the algorithm for 1000 iterations and noting the norms of $\mathfrak{b}$. For all the fields that we considered, it turned out that the actual ideals $\mathfrak{b}$ that are generated have norms which are much less than the upper bound. Since the norms are signficantly lower, we decided to check the smoothness probabilities $\mathcal{P}(x, y)$ where x is the maximum norm that was experimentally observed and y is equal to C. It turned out that these smoothness probabilities are sufficiently high for relation collection to proceed.

In Table 4, for the various fields considered in this work, we provide the values of the upper bound $\mathfrak{B}_\beta$ on the norm of $\mathfrak{b}$ for $\beta = 2$ and $\beta = \beta_{th}$, the corresponding smoothness probabilities $\mathcal{P}(\mathfrak{B}_\beta, B)$, the experimentally observed maximum norms (denoted as $\mathfrak{M}$) of the ideal $\mathfrak{b}$ and the corresponding smoothness probabilities $\mathcal{P}(\mathfrak{M}, C)$. The values of B used in the computation of $\mathcal{P}(\mathfrak{B}_\beta, B)$ are from Table 3, which corresponds to $c = 1$. As discussed earlier, by suitably choosing the value of c, the value of B can be made close to the value of C. The corresponding smoothness probabilities $\mathcal{P}(\mathfrak{B}_\beta, C)$ are very low. If the norms of the ideals $\mathfrak{b}$ are indeed close to the upper bound $\mathfrak{B}_\beta$, then using C as the smoothness bound will make relation collection very inefficient.

From Table 4, it may be noted that the value of $\mathfrak{M}$ is substantially lower than the value of $\mathfrak{B}_\beta$ for both $\beta = 2$ and $\beta = \beta_{th}$. For $n = 10$ and $n = 15$, the value of $\mathfrak{M}$ is close to the value of C so that $\mathcal{P}(\mathfrak{M}, C)$ is very close to one. For $n = 20$ and $n = 25$, the value of $\mathcal{P}(\mathfrak{M}, C)$ is close to the value of $\mathcal{P}(\mathfrak{B}_{\beta_{th}}, B)$. Based on the values in Table 4, we may conclude that if relation collection is done with the factor basis as the prime ideals whose norms are below Bach's bound, then in practice, the smoothness probability is sufficiently high for relation collection to be possible. Due to this, we decided to proceed with C as the smoothness bound in Algorithm 1 instead of B which is obtained from Table 3. Note that this conclusion is for the number fields that have been considered in this work. Whether such an observation would hold in general for larger number fields is not clear. Nonetheless, our experiments indicate that even for larger number fields, it would be worthwhile to experimentally obtain $\mathfrak{M}$ and then decide whether B can be taken to equal to C, or, whether it needs to be chosen to be greater than C, and if so, how much larger. Using the upper bound on the norm of $\mathfrak{b}$ as the guideline for choosing B (as has been done in the asymptotic analysis [9]), may lead to much larger factor basis and a significantly less efficient algorithm.

For the number fields that we have chosen, it has turned out that using C as the smoothness bound is sufficient in practice. As noted earlier, in asymptotic terms B is much larger than C. So, for larger fields, choosing B to be equal to C will perhaps not be appropriate. Further experiments with larger fields are required to determine how much larger should B be compared to C.

4.3 Fixation of Parameters for Algorithms 1 and 3

The parameters k and A need to be fixed for Algorithm 1. Gélin [9] specifies these two parameters to be $\operatorname{poly}\log|\Delta_{\mathcal{K}}|$. In practice, we need concrete values of these parameters. As the values of A and k increase, the norms of the ideals $\mathfrak{a}$ in Algorithm 1 also increase. This causes certain difficulties in the Magma implementation. The lattice $\sigma(\mathfrak{a})$ needs to be constructed and then the BKZ reduction needs to be performed on this lattice. Both of these steps crucially depend on the value of the precision used by Magma for the computations. From the experiments, we have the following observations regarding the issue of precision.

1. For a particular value of the precision, lattice construction is possible for ideals having norms below a certain bound. We could not, however, determine the relationship between the precision and the bound on norm. The general observation we have is that by increasing precision, it becomes possible to perform lattice construction for ideals with larger norms.

2. For the BKZ reduction to be possible, Magma requires the gram matrix to be positive definite. The check for positive definiteness requires a higher precision than the precision required for the lattice construction.

3. Increasing precision slows down the computations.

In view of the above issues, choosing high values of A and k lead to abnormal termination of the Magma programs. So, we set A to be equal to 2 and did some experiments with varying precision to determine a suitable value of k. Finally, we set $k = 15$. The corresponding precision for lattice construction was fixed to be 2000 and the precision for the positive definiteness check on the gram matrix was set to be 6000.

For Algorithm 3, the values of the parameters κ_0 and λ need to be determined. Additionally, the values of κ and R which determine the size of the table $\mathfrak{T}$ also need to be fixed. Note that the number of rows in $\mathfrak{T}$ is about R/κ times N_b. We have chosen $R = 2$ and $\kappa = 4$ so that the number of pre-computed ideals stored in $\mathfrak{T}$ is about half of N_b. The value κ_0 has been set to 2 and we have considered the start points of the walks to be products of two ideals in $\mathfrak{C}$. Recall that based on our experiments, we have fixed the factor basis itself to be $\mathcal{C}$ and so using products of a pair of ideals in $\mathcal{C}$ provides sufficiently many walks. As mentioned earlier, the value of λ needs to be fixed so as to ensure that the maximum norm of the ideals visited by a walk is around the same value as the norms of the ideals $\mathfrak{a}$ generated in Algorithm 1. Since, we fixed k to be equal to 15, we fixed $\lambda = 8$. We conducted several experiments to confirm that with $\lambda = 8$, there was no failure in either the lattice construction, or the BKZ-reduction. The precision used for the lattice construction was 2000, which is the same as that used for Algorithm 1. Since the initial experiments indicated that there is no failure when $\lambda = 8$, we did not perform an explicit check for the positive definiteness of the gram matrix.

The hash function $\mathcal{H}$ used to access entries of the pre-computed table $\mathfrak{T}$ should ensure a more or less uniform distribution over the entries of the table. In our implementations, we have used the in-built hash function of Magma reduced modulo m (the size of the pre-computed table) to instantiate $\mathcal{H}$. We have also experimented with other hash functions defined using simple arithmetic. There was no significant change in the results.

4.4 Experimental Set-Up and Timing Results

All our computations were done on a server which has the configuration of Intel Xeon E7-8890 @ 2.50 GHz with 72 physical cores and 144 logical cores. As mentioned earlier, the computations were done using Magma version V2.22-3. We conducted two sets of experiments.

The goal of the first set of experiments is to compare the performances of Algorithms 1 and 3. Since we have set the smoothness bound B to be equal to C, the number of ideals in the factor basis is N_b. For the four fields, we generated the factor basis and determined N_b. The values of N_b corresponding to the fields are shown in Table 4. The number of relations required is a little more than the size of the factor

field	Algorithm 1		Algorithm 3		
	# iter	time (t_1)	# iter	time (t_3)	t_1/t_3
$\mathcal{K}_1$	3004	876	3385	266	3.29
$\mathcal{K}_2$	5007	4024	5642	1405	2.86
$\mathcal{K}_3$	8370	17702	9285	5903	3.00
$\mathcal{K}_4$	165840	717386	183631	211381	3.39

Table 5: Comparison of Algorithms 1 and 3.

Algorithm	$\mathcal{K}_1$	$\mathcal{K}_2$	$\mathcal{K}_3$	$\mathcal{K}_4$
Algorithm 1	≈ 1.00	≈ 1.00	≈ 0.60	≈ 0.03
Algorithm 3	≈ 0.89	≈ 0.89	≈ 0.54	≈ 0.03

Table 6: Empirical estimates of the smoothness probabilities for Algorithms 1 and 3 obtained from Table 5.

basis. Since $N_{\mathfrak{b}}$ is quite small for $\mathcal{K}_1$ and $\mathcal{K}_2$, we decided to generate a complete set of relations for these two fields using both Algorithms 1 and 3; for $\mathcal{K}_1$, we generated 3000 relations while for $\mathcal{K}_2$, we generated 5000 relations. In comparison, for $\mathcal{K}_3$ and $\mathcal{K}_4$, $N_{\mathfrak{b}}$ is a little higher, so, for these two fields we decided to compare the performances of the two algorithms for collecting 5000 relations. For each of the experiments, 25 processes were run in parallel. For $\mathcal{K}_1$, each of the processes was tasked with collecting 120 relations, while for the other three fields, each of the processes was tasked with collecting 200 relations. For all of the fields, all the relations obtained by both Algorithms 1 and 3 were distinct. In Table 7, we provide the total number of iterations and the total times (in seconds) required by all the processes for both the algorithms.

From Table 5, it is to be noted that the total number of iterations required by Algorithm 3 is slightly more than that required by Algorithm 1. Recall that in both Algorithms 1 and 3, a relation is obtained whenever the ideal $\mathfrak{b}$ is smooth over the factor basis. An important parameter in the assessment of the complexity is the probability of smoothness. Empirically, the total number of relations collected divided by the total number of iterations required is an estimate of the smoothness probability of $\mathfrak{b}$. These estimates are shown in Table 6. The probabilities corresponding to Algorithm 1 are slightly higher. It is perhaps useful to compare these probabilities with the probability estimates given in Table 4. While for $\mathcal{K}_1$ and $\mathcal{K}_2$, the empirical estimates in Table 6 are close to $\mathcal{P}(\mathfrak{M}, C)$, for $\mathcal{K}_3$ and $\mathcal{K}_4$, the empirical estimates in Table 6 are substantially larger. The reason for this is the fact that in the computation $\mathcal{P}(\mathfrak{M}, C)$, $\mathfrak{M}$ has been taken to be the maximum of the norms of the ideals $\mathfrak{b}$ generated in about 1000 trials. The average of the norms is substantially lower than the maximum, so that $\mathcal{P}(\mathfrak{M}, C)$ is a substantial underestimate of the probability of smoothness of $\mathfrak{b}$.

The main point to observe from Table 5 is the ratio t_1/t_3. This figure varies from 2.86 to 3.39 indicating that in practice, Algorithm 3 is about three times faster than Algorithm 1. The speed-up factor of three is determined by the choice of the parameters for Algorithms 1 and 3 mentioned earlier. A different choice of parameters may lead to a different speed-up factor. As explained earlier, the main advantage of Algorithm 3 over Algorithm 1 is the reduction in the number of ideal multiplications in the generation of the ideal $\mathfrak{a}$. The main point of the experiments is to show that this advantage can indeed be realised in practice.

The first set of experiments provide evidence that in practice Algorithm 3 peforms better than Algorithm 1. A second set of experiments was conducted to demonstrate that Algorithm 3 can indeed generate the full set of relations for reasonable size number fields. To this end, we used Algorithm 3 to collect 15000 relations for $\mathcal{K}_3$ and 35000 relations for $\mathcal{K}_4$. For $\mathcal{K}_3$, we ran 75 processes each tasked with collecting 200 relations while for $\mathcal{K}_4$, we ran 70 processes each tasked with collecting 500 relations. In the case of $\mathcal{K}_3$ all the obtained relations were distinct, while for $\mathcal{K}_4$, 34994 relations were distinct. The total number

field	# iter	time
$\mathcal{K}_3$	28162	22937
$\mathcal{K}_4$	1268434	1865071

Table 7: Number of iterations and times (in seconds) to collect 15000 relations for $\mathcal{K}_3$ and 35000 relations for $\mathcal{K}_4$ using Algorithm 3.

of iterations and the total times required by all the processes for these experiments are shown in Table 7.

From Table 5, we note that the estimated probabilities of smoothness of the ideal $\mathfrak{b}$ (i.e., the number of relations divided by the number of iterations) are 0.53 and 0.03 for $\mathcal{K}_3$ and $\mathcal{K}_4$ respectively. These are close to the probability estimates given in Table 6 for Algorithm 3. The main point of the second set of experiments is to show that Algorithm 3 can indeed be used to generate a complete set of relations.

The time to generate the pre-computed table $\mathfrak{T}$ has not been considered in either Table 5 or 7. We would like to highlight that for a reasonable size number field, the time to generate the full set of relations is substantially higher than the time to generate $\mathfrak{T}$. Instead of comparing the times, we compare the number of ideal multiplications required for the two tasks in the case of $\mathcal{K}_4$. From Table 7, the number of iterations required to collect 35000 relations is 1268434. So, Algorithm 3 considers this number of ideals $\mathfrak{a}$. The generation of the start ideal of each walk requires a single ideal multiplication and each step of the walk also requires a single ideal multiplication. So, the number of ideal multiplications to generate 1268434 ideals is also 1268434. Now, consider the generation of $\mathfrak{T}$. The number of entries in $\mathfrak{T}$ is $16192 = q \times R$, where $q = 8096$ and $R = 2$. Of the 16192 ideals in $\mathfrak{T}$, 16190 ideals are products of 4 primes ideals from $\mathcal{C}$, while 2 ideals are products of 5 prime ideals from $\mathcal{C}$. So, the total number of ideal multiplications required to generate the ideals in $\mathfrak{T}$ $16190 \times 3 + 2 \times 4 = 48578$, which is about 3.8% of the number of ideal multiplications required to generate all the ideals. Consequently, even if the time for the generation of $\mathfrak{T}$ is taken into consideration, Algorithm 3 will perform better than Algorithm 1 for collecting a complete set of relations for a large enough number field.

Simple Setting of Parameters. For the previously mentioned experiments, the walk length in Algorithm 3 was set to 8 and the start points were products of two ideals in $\mathcal{C}$. Given that the concrete number fields are not too large, we decided to experiment with relation collection using Algorithm 3 with walk length 1. This means that each walk visits only one ideal which is the start point, where each start point is a product of two ideals. With this setting, Algorithm 3 essentially becomes the same as Algorithm 1 with $k = 2$ and $A = 1$. Since the setting was simple, we also reduced the precision to the default precision for Magma which is 167. We wished to find out whether such a basic setting is adequate for collecting relations.

We ran the relation collection processes for $\mathcal{K}_3$ and $\mathcal{K}_4$ as before with the goal of collecting 15000 and 35000 relations respectively. The number of iterations required were 24901 and 970114; and the number of distinct relations obtained were 14999 and 295533 respectively. The number of distinct relations obtained for $\mathcal{K}_3$ was adequate, but for $\mathcal{K}_4$ there was a sharp drop from the target of 35000 relations. The required times for $\mathcal{K}_3$ and $\mathcal{K}_4$ were 5419 and 295533 seconds respectively. Note that compared to the times in Table 7, there is a marked improvement in the times.

So, for the number fields that have been considered, the above mentioned simple setting leads to improved times, though for $\mathcal{K}_4$ the number of distinct relations obtained are not sufficient and more relations would need to be obtained to make up the deficit. For larger number fields, however, it is unlikely that such a simple setting would be sufficient for generating the required relations. A non-trivial walk length would be required in Algorithm 3 to be able to explore a larger portion of the ideal space.

5 Conclusion

In this paper, we have introduced a technique to perform a pseudo-random walk over ideals. After the first step, each step of the walk requires a single ideal multiplication. The ideals visited by the walk are used for relation collection in exactly the same manner as used in Gélin's algorithm [9]. The practical advantage over Gélin's algorithm is the reduction in the number of ideal multiplications required to generate the next ideal to be tested. Our Magma implementations of both the new algorithm and Gélin's algorithm confirm that there is indeed a practical speed-up.

Acknowledgement

We would like to thank Alexandre Gélin for various discussions regarding class group and its computational aspects. We also appreciate the help provided by Allan Steel, Claus Fieker, Geoffrey Bailey, Jean-François Biasse, John Cannon for computations with lattices using Magma.

References

[1] Eric Bach. Explicit bounds for primality testing and related problems. *Mathematics of Computation*, 55(191):355–380, 1990.

[2] 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.

[3] 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.

[4] Dan Boneh, Benedikt Bünz, and Ben Fisch. Batching techniques for accumulators with applications to iops and stateless blockchains. In Alexandra Boldyreva and Daniele Micciancio, editors, *Advances in Cryptology - CRYPTO 2019 - 39th Annual International Cryptology Conference, Santa Barbara, CA, USA, August 18-22, 2019, Proceedings, Part I*, volume 11692 of *Lecture Notes in Computer Science*, pages 561–586. Springer, 2019.

[5] Johannes Buchmann. A subexponential algorithm for the determination of class groups and regulators of algebraic number fields. *Séminaire de théorie des nombres, Paris*, 1989(1990):27–41, 1988.

[6] Benedikt Bünz, Ben Fisch, and Alan Szepieniec. Transparent snarks from DARK compilers. In Anne Canteaut and Yuval Ishai, editors, *Advances in Cryptology - EUROCRYPT 2020 - 39th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Zagreb, Croatia, May 10-14, 2020, Proceedings, Part I*, volume 12105 of *Lecture Notes in Computer Science*, pages 677–706. Springer, 2020.

[7] Henri Cohen. A course in computational algebraic number theory. *Graduate texts in Math.*, 138:88, 1993.

[8] 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.

[9] Alexandre Gélin. On the complexity of class group computations for large degree number fields. *arXiv preprint arXiv:1810.11396*, 2018.

[10] Alexandre Gélin. Reducing the complexity for class group computations using small defining polynomials. *arXiv preprint arXiv:1810.12010*, 2018.

[11] 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.

[12] 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.

[13] James L Hafner and Kevin S McCurley. A rigorous subexponential algorithm for computation of class groups. *Journal of the American mathematical society*, 2(4):839–850, 1989.

[14] John W Jones and David P Roberts. A database of number fields. *LMS Journal of Computation and Mathematics*, 17(1):595–618, 2014.

[15] Daniel Shanks. Class number, a theory of factorization, and genera. In *Proc. of Symp. Math. Soc., 1969*, volume 20, pages 415–440, 1969.

[16] Daniel Shanks. The infrastructure of a real quadratic field and its applications. In *Proceedings of the 1972 Number Theory Conference*, pages 217–224, 1972.