Practical improvements to class group and regulator computation of real quadratic fields

Jean-François Biasse, Jacobson Michael

Practical improvements to class group and regulator computation of real quadratic fields

Jean-François Biasse[1] and Michael J. Jacobson, Jr.[2⋆]

[1] École Polytechnique, 91128 Palaiseau, France
biasse@lix.polytechnique.fr
[2] Department of Computer Science, University of Calgary
2500 University Drive NW, Calgary, Alberta, Canada T2N 1N4
jacobs@cpsc.ucalgary.ca

Abstract. We present improvements to the index-calculus algorithm for the computation of the ideal class group and regulator of a real quadratic field. Our improvements consist of applying the double large prime strategy, an improved structured Gaussian elimination strategy, and the use of Bernstein's batch smoothness algorithm. We achieve a significant speed-up and are able to compute the ideal class group structure and the regulator corresponding to a number field with a 110-decimal digit discriminant.

1 Introduction

Computing invariants of real quadratic fields, in particular the ideal class group and the regulator, has been of interest since the time of Gauss, and today has a variety of applications. For example, solving the well-known Pell equation is intimately linked to computing the regulator, and integer factorization algorithms have been developed that make use of this invariant. Public-key cryptosystems have also been developed whose security is related to the presumed difficulty of these computational tasks. See [16] for details.

The fastest algorithm for computing the ideal class group and regulator in practice is a variation of Buchmann's index-calculus algorithm [6] due to Jacobson [14]. The algorithm on which it is based has subexponential complexity in the size of the discriminant of the field. The version in [14] includes several practical enhancements, including the use of self-initialized sieving to generate relations, a single large-prime variant (based on that of Buchmann and Düllman [7] in the case of imaginary quadratic fields), and a practical version of the required linear algebra. This approach proved to work well, enabling the computation of the ideal class group and regulator of a real quadratic field with a 101-decimal digit discriminant [15]. Unfortunately, both the complexity results of Buchmann's algorithm and the correctness of the output are dependent on the Generalized Riemann Hypothesis (GRH). Nevertheless, for fields with large discriminants, this approach is the only one that works.

⋆ The second author is supported in part by NSERC of Canada.

Recently, Biasse [4] presented practical improvements to the corresponding algorithm for imaginary quadratic fields. These included a double large prime variant and improved algorithms for the required linear algebra. The resulting algorithm was indeed faster then the previous state-of-the-art [14], and enabled the computation of the ideal class group of an imaginary quadratic field with 110 decimal digit discriminant.

In this paper, we describe a number of practical improvements to the index-calculus algorithm for computing the class group and regulator of a real quadratic field. In addition to adaptations of Biasse's improvements in the imaginary case, we have found some modifications designed to improve the regulator computation part of the algorithm. We also investigate applying an idea of Bernstein [3] to factor residues produced by the sieve using a batch smoothness test. Extensive computations demonstrating the effectiveness of our improvements are presented, including the computation of class group and regulator of a real quadratic field with 110 decimal digit discriminant.

This paper is organized as follows. In the next section, we briefly recall the required background of real quadratic fields, and give an overview of the index-calculus algorithm using self-initialized sieving. Our improvements to the algorithm are described in Section 3, followed by numerical results in Section 4.

2 Real Quadratic Fields

We present an overview of required concepts related to real quadratic fields and the index-calculus algorithm for computing invariants. For more details, see [16].

Let $K = \mathbb{Q}(\sqrt{\Delta})$ be the real quadratic field of discriminant Δ, where Δ is a positive integer congruent to 0 or 1 modulo 4 with Δ or $\Delta/4$ square-free. The integral closure of $\mathbb{Z}$ in K, called the maximal order, is denoted by $\mathcal{O}_\Delta$. An interesting aspect of real quadratic fields is that their maximal orders contain infinitely many non-trivial units, i.e., units that are not roots of unity. More precisely, the unit group of $\mathcal{O}_\Delta$ consists of an order 2 torsion subgroup and an infinite cyclic group. The smallest unit greater than 1, denoted by ε_Δ, is called the fundamental unit. The regulator of $\mathcal{O}_\Delta$ is defined as $R_\Delta = \log \varepsilon_\Delta$.

The fractional ideals of K play an important role in the index-calculus algorithm described in this paper. In our setting, a fractional ideal is a rank 2 $\mathbb{Z}$-submodule of K. Any fractional ideal can be represented as

$$\mathfrak{a} = \frac{s}{d}\left[a\mathbb{Z} + \frac{b+\sqrt{\Delta}}{2}\mathbb{Z}\right] \ ,$$

where $a, b, s, d \in \mathbb{Z}$ and $4a \mid b^2 - \Delta$. The integers a, s, and d are unique, and b is defined modulo $2a$. The ideal $\mathfrak{a}$ is said to be primitive if $s = 1$, and $d\mathfrak{a} \subseteq \mathcal{O}_\Delta$ is integral. The norm of $\mathfrak{a}$ is given by $\mathcal{N}(\mathfrak{a}) = as^2/d^2$.

Ideals can be multiplied using Gauss's composition formulas for indefinite binary quadratic forms. Ideal norm respects ideal multiplication, and the set

$\mathcal{I}_\Delta$ forms an infinite abelian group with identity $\mathcal{O}_\Delta$ under this operation. The inverse of $\mathfrak{a}$ is

$$\mathfrak{a}^{-1} = \frac{d}{sa}\left[a\mathbb{Z} + \frac{-b+\sqrt{\Delta}}{2}\mathbb{Z}\right] \quad .$$

The group $\mathcal{I}_\Delta$ is generated by the prime ideals of $\mathcal{O}_\Delta$, namely those integral ideals of the form $p\mathbb{Z}+(b_p+\sqrt{\Delta})/2\mathbb{Z}$ where p is a prime that is split or ramified in K. As $\mathcal{O}_\Delta$ is a Dedekind domain, the integral part of any fractional ideal can be factored uniquely as a product of prime ideals. To factor $\mathfrak{a}$, it suffices to factor $\mathcal{N}(\mathfrak{a})$ and, for each prime p dividing the norm, determine whether the prime ideal $\mathfrak{p}$ or $\mathfrak{p}^{-1}$ divides $\mathfrak{a}$ according to whether $b \equiv b_p$ or $-b_p$ modulo $2p$.

The ideal class group, denoted by Cl_Δ, is the factor group $\mathcal{I}_\Delta/\mathcal{P}_\Delta$, where $\mathcal{P}_\Delta \subseteq \mathcal{I}_\Delta$ is the subgroup of principal ideals. The class group is finite abelian, and its order is called the class number, denoted by h_Δ. By computing the class group we mean computing the elementary divisors $m_1, \ldots, m_l$ with $m_{i+1} \mid m_i$ for $1 \leq i < l$ such that $Cl_\Delta \cong \mathbb{Z}/m_1\mathbb{Z} \times \cdots \times \mathbb{Z}/m_l\mathbb{Z}$.

2.1 The Index-Calculus Algorithm

Like other index-calculus algorithms, the algorithm for computing the class group and regulator relies on finding certain smooth quantities, those whose prime divisors are all small in some sense. In the case of quadratic fields, one searches for smooth principal ideals for which all prime ideal divisors have norm less than a given bound B_1. The set of prime ideals $\mathcal{B} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_n\}$ with $\mathfrak{p}_i \leq B_1$ is called the factor base.

A principal ideal $(\alpha) = \mathfrak{p}_1^{e_1} \ldots \mathfrak{p}_n^{e_n}$ with $\alpha \in K$ that factors completely over the factor base yields the relation $(e_1, \ldots, e_n, \log|\alpha|)$. The key to the index-calculus algorithm is the fact, proved by Buchmann [6], that the set of all relations forms a sublattice $\Lambda \subset \mathbb{Z}^n \times \mathbb{R}$ of determinant $h_\Delta R_\Delta$ provided that the prime ideals in the factor base generate Cl_Δ. This follows, in part, due to the fact that L, the integer component of Λ, is the kernel of the homomorphism from $\mathbb{Z}^n$ to Cl_Δ given by $\mathfrak{p}_1^{e_1} \ldots \mathfrak{p}_n^{e_n}$ for $(e_1, \ldots, e_n) \in \mathbb{Z}^n$. If $\mathfrak{p}_1, \ldots, \mathfrak{p}_n$ generate Cl_Δ, then this homomorphism is surjective, and the homomorphism theorem then implies that $\mathbb{Z}^n/L \cong Cl_\Delta$.

The main idea behind the index-calculus algorithm is to find random relations until they generate the entire relation lattice Λ. Let Λ' denote the sublattice of Λ generated by the relations that have been computed. To determine whether $\Lambda' = \Lambda$, one computes an approximation h^* of $h_\Delta R_\Delta$ such that $h^* < h_\Delta R_\Delta < 2h^*$. The value h^* is obtained by approximating the L-function $L(1, \chi_\Delta)$, where χ_Δ denotes the Kronecker symbol (Δ/p), and applying the analytic class number formula. If $\Lambda' \subset \Lambda$, then $\det(\Lambda')$ is a integer multiple of $h_\Delta R_\Delta$. Thus, $\Lambda' = \Lambda$ as soon as $\det(\Lambda') < 2h^*$, because $h_\Delta R_\Delta$ is the only integer multiple of itself in the interval $(h^*, 2h^*)$.

As described in [14], an adaptation of the strategy used in the self-initialized quadratic sieve (SIQS) factoring algorithm is used to compute relations. First,

compute the ideal $\mathfrak{a} = \mathfrak{p}_1^{e_1} \dots \mathfrak{p}_n^{e_n} = (1/d)[a\mathbb{Z} + (b + \sqrt{\Delta})/2\mathbb{Z}]$ with $\mathcal{N}(\mathfrak{a}) = a/d^2$. Let $\alpha = (ax + (b + \sqrt{\Delta})/2y)/d$ with $x, y \in \mathbb{Z}$ be an arbitrary element in $\mathfrak{a}$. Then

$$\mathcal{N}(\alpha) = \frac{1}{d^2}\left(ax + \frac{b+\sqrt{\Delta}}{2}y\right)\left(ax + \frac{b-\sqrt{\Delta}}{2}y\right) = (a/d^2)(ax^2 + bxy + cy^2)$$

where $c = (b^2 - \Delta)/(4a)$. Because ideal norm is multiplicative, there exists an ideal $\mathfrak{b}$ with $\mathcal{N}(\mathfrak{b}) = ax^2 + bxy + cy^2$ such that $(\alpha) = \mathfrak{a}\mathfrak{b}$. Thus, finding x and y such that $\mathcal{N}(\mathfrak{b})$ factors over the norms of the prime ideals in the factor base yields a relation. Such x and y can be found by sieving the polynomial $\varphi(x, y) = ax^2 + bxy + cy^2$, and a careful selection of the ideals $\mathfrak{a}$ yields a generalization of self-initialization, in which the coefficients of the sieving polynomials and their roots modulo the prime ideal norms can be computed quickly. In practice, we use $\varphi(x, 1)$ for sieving, so that the algorithm resembles the SIQS more closely. For more details, see [14] or [16].

The determinant of the relation lattice Λ' is computed in two stages. The first step is to compute the determinant of the integer part of this sublattice by finding a basis in Hermite normal form (HNF). Once Λ' has full rank, the determinant of this basis is computed as the product of the diagonal elements in a matrix representation of the basis vectors. The group structure is then computed by finding the Smith normal form of this matrix. The real part of $\det(\Lambda')$, a multiple of the regulator R_Δ, is computed by first finding a basis of the kernel of the matrix consisting of the integer parts of the relations. Every vector $(k_1, \dots, k_m) \in \mathbb{Z}^m$ in the kernel corresponds to a multiple of the regulator computed with $mR_\Delta = k_1 \log|\alpha_1| + \dots + k_m \log|\alpha_m|$. The "real gcd" of the multiples $m_1 R_\Delta, \dots, m_n R_\Delta$ computed from each basis vector of the kernel, defined as $\gcd(m_1, \dots, m_n)R_\Delta$, is then the real part of $\det(\Lambda')$. An algorithm of Maurer [21] can be used to compute the real gcd efficiently and with guaranteed numerical accuracy given explicit representations of the α_i and the kernel vectors.

As mentioned in the introduction, the correctness of this algorithm depends on the truth of the Generalized Riemann Hypothesis. In fact, the GRH must be invoked in two places. The first is to compute a sufficiently accurate approximation h^* of $h_\Delta R_\Delta$ via a method due to Bach [2]. Without the GRH, an exponential number of terms in the Euler product used to approximate $L(1, \chi_\Delta)$ must be used (see, for example, [20]). The second is to ensure that the factor base generates Cl_Δ. Without the GRH, an exponential size factor base is required, whereas by a theorem of Bach [1] the prime ideals of norm less than $6\log(\Delta)^2$ suffice. In practice, an even smaller factor base is often used, but in that case, the factor base must be verified by showing that every remaining prime ideal with norm less than Bach's bound can be factored over the ideals in the factor base.

3 Practical Improvements

In this section, we describe our practical improvements for computing the class group structure and the regulator of a the real quadratic field. Some of these improvements, such as the double large prime variant and structured Gaussian elimination, were used in [4] for the simpler case of imaginary quadratic number fields. On the other hand, the batch smoothness test and system solving based methods for computing the regulator had never been implemented in the context of number fields before.

3.1 Relation collection

Improving the relation collection phase allows us to speed up every other stage of the algorithm. Indeed, the faster the relations are found, the smaller the factor base can be, thus reducing the dimensions of the relation matrix and the time taken by the linear algebra phase. In addition, the verification phase also relies on our ability to find relations and therefore benefits from improvements to the relation collection phase. Throughout the rest of the paper, M denotes the relation matrix, the matrix whose rows are the integer parts of the relations.

Large prime variants The large prime variants were developed in the context of integer factorization to speed up the relation collection phase in both the quadratic sieve and the number field sieve. A single large prime variant was described by Buchmann and Düllman [7] for computing the class group of an imaginary quadratic field, and adapted to the real case by Jacobson [14]. Biasse [4] described how the double large prime strategy could be using in the imaginary case, and obtained a significant speed-up.

The idea is to keep relations involving one or two extra primes not in the factor base of norm less than $B_2 \geq B_1$. These relations thus have the form

$$(\alpha) = \mathfrak{p}_1^{e_1} \dots \mathfrak{p}_n^{e_n} \mathfrak{p} \quad \text{and} \quad (\alpha) = \mathfrak{p}_1^{e_1} \dots \mathfrak{p}_n^{e_n} \mathfrak{p}\mathfrak{p}'$$

for $\mathfrak{p}_i$ in $\mathcal{B}$, and for $\mathfrak{p}, \mathfrak{p}'$ of norm less than B_2. We will refer to these types of partial relations as 1-partial relations and 2-partial relations, respectively. Keeping partial relations only involving one large prime is the single large prime variant, whereas keeping those involving one or two is the double large prime variant which was first described by Lenstra and Manasse [17]. We do not consider the case of more large primes, but it is a possibility that has been studied in the context of factorization [10].

Partial relations may be identified as follows. Let m be the remainder of $\varphi(x, 1)$ after the division by all primes $p \leq B_1$, and assume that $B_2 < B_1^2$. If $m = 1$ then we have a full relation. If $m \leq B_2$ then we have a 1-partial relation. We can see here that detecting 1-partial relations is almost for free. If we also intend to collect 2-partial relations then we have to consider the following possibilities:

1. $m > B_2^2$;
2. m is prime and $m > B_2$;
3. m is prime and $m \leq B_2$;
4. m is composite and $B_1^2 < m \leq B_2^2$.

In Cases 1 and 2 we discard the relation. In Case 3 we have a 1-partial relation, and in Case 4 we have $m = pp'$ where $p = \mathcal{N}(\mathfrak{p})$ and $p' = \mathcal{N}(\mathfrak{p}')$. Cases 1, 2, and 3 can be checked very easily, but if none are satisfied we need to factor m in order to determine whether Case 4 is satisfied. We used Milan's implementation of the SQUFOF algorithm [22] based on the theoretical work of [12] to factor the m values produced.

Even though we might have to factor the remainder, partial relations are found much faster than full relations. However, the dimensions of the resulting matrix are much larger, thus preventing us from running the linear algebra phase directly on the resulting relation matrix. In addition, we have to find many more relations since we have to produce a full rank matrix. We will see in §3.2 how to reduce the dimensions of the relation matrix using Gaussian elimination techniques.

Batch smoothness test After detecting potential candidates for smooth integers via the SIQS, one has to certify their smoothness. In [4, 14], this was done by trial division with the primes in the factor base. The time taken by trial division can be shortened by using Bernstein's batch smoothness test [3], which uses a product tree structure and modular arithmetic to factor a batch of residues simultaneously in time $O\left(b(\log b)^2 \log\log b\right)$ where b is the total number of input bits.

Instead of testing the smoothness of every potential candidate as soon as they are discovered, we rather stored them and tested them at the same time using Bernstein's method as soon their number exceeded a certain limit. This improvement has an effect that is all the more important when the time spent in the trial division is long. In our algorithm, this time mostly depends on the tolerance value T, a parameter used to control the number of candidates yielded by the sieve for smoothness testing.

3.2 Structured Gaussian Elimination

As mentioned in §2.1, in order to determine whether the computed relations generate the entire relation lattice, we need to compute the HNF basis of the sublattice they generate. This can be done by putting the integer components of the relations as rows in a relation matrix, and computing the HNF.

The first step when using large primes is to compute full relations from all of the partial relations. Traditionally, rows were recombined to give full relations as follows. In the case of 1-partial relations, any pair of relations involving the same large prime $\mathfrak{p}$ were recombined into a full relation. In the case of 2-partial relations, Lenstra [17] described the construction of a graph whose vertices were the relations and whose edges linked vertices having one large prime in common.

Finding independent cycles in this graph allows us to recombine partial relations into full relations.

In this paper, we instead follow the approach of Cavallar [8], developed for the number field sieve, and adapted by the first author to the computation of ideal class group structures in imaginary quadratic number fields [4], which uses Gaussian elimination on columns. The ideas is to eliminate columns using structured Gaussian strategies until the dimensions of the matrix are small enough to allow the computation of the HNF with standard algorithms.

Let us recall a few definitions. First, subtracting two rows is called *merging*. If two relations corresponding to rows r_1 and r_2 share the same prime $\mathfrak{p}$ with coefficients c_1 and c_2 respectively, then multiplying r_1 by c_2 and r_2 by c_1 and merging is called *pivoting*. Finally, finding a sequence of pivots leading to the elimination of a column of Hamming weight k is a k-way merge.

We aim to reduce the dimensions of the relation matrix by performing k-way merges on the columns of weight $k = 1, \ldots, w$ in increasing order for a certain bound w. To limit the growth of the density and of the size of the coefficients induced by these operations, we used optimized pivoting strategies. In what follows we describe an algorithm performing k-way merges to minimize the growth of both the density and the size of the coefficients, thus allowing us to go deeper in the elimination process and delay the explosion of the coefficients.

As in [4], we define a cost function C mapping rows onto the integers. The one used in [4] satisfied

$$C(r) = \sum_{1 \leq |e_i| \leq Q} 1 + c \sum_{|e_j| > Q} 1, \tag{1}$$

where c and Q are positive numbers, and $r = [e_1, \ldots, e_n]$ is a row corresponding to $(\alpha) = \prod_i \mathfrak{p}_i^{e_i}$. This way, the heaviest rows are those which have a high density and large coefficients. In our experiments for this work, we used a different cost function, see §4.1. Then, to perform a k-way merge on a given column, we construct a complete graph $\mathcal{G}$ of size k such that

- the vertices are the rows r_i, and
- every edge linking r_i and r_j has weight $C(r_{ij})$, where r_{ij} is obtained by pivoting r_i and r_j.

Finding the best sequence of pivots with respect to the chosen cost function C is equivalent to finding the minimum spanning tree $\mathcal{T}$ of $\mathcal{G}$, and then recombining every row r with its parent starting with the leaves of $\mathcal{T}$.

Unlike in [4], we need to keep track of the permutations we apply to the relation matrix, and of the empty columns representing primes of norm less than $6 \log^2 \Delta$. This will be required for the regulator computation part of the algorithm described next.

3.3 Regulator computation

As mentioned in §2.1, the usual way to compute the regulator is to find a basis of the kernel of the relation matrix, compute integer multiples of the regulator

from these basis vectors, and compute their real gcd using Maurer's algorithm [21]. If $\det \Lambda' > 2h^*$, then either the class number or regulator computed is too large, and we need to find extra relations corresponding to new generators, and new kernel vectors involving them.

In this section, we describe a way of taking advantage of the large number of generators involved in the different partial relations. Indeed, the dimensions of the relation matrix before the Gaussian elimination stage is much larger than in the base scenario and thus involves more generators. Consequently, given a set of $k \leq \dim(\ker M)$ kernel vectors $(u_1^j, \ldots, u_n^j)_{j \leq k}$, the probability that the corresponding elements

$$v_j := u_1^j \log |\alpha_1| + \ldots + u_n^j \log |\alpha_n| \ ,$$

where α_i is the generator of the i-th relation, can be recombined into R is much larger. On the other hand, the dimensions of the matrix prevents us from running a kernel computation directly after the relation collection phase. Thus, rather than attempting to compute the kernel, we use a method similar to that of Vollmer [24] based on solving linear systems.

The first step of our algorithm consists of putting the matrix in a pseudo-lower triangular form using a permutation obtained during the Gaussian elimination phase. Indeed, as part of this computation we obtain a unimodular matrix $U \in \mathbb{Z}^{n \times n}$ such that

$$UM = \left(\begin{array}{c:cc} A & (0) & \\ \hdashline & 1 & (0) \\ (*) & \ddots & \\ & (*) & 1 \end{array} \right) .$$

Thus, solving a linear system of the form $xM = b$ for a vector $b \in \mathbb{Z}^m$ boils down to solving a system of the form $x'A = b'$, then doing a trivial descent through the diagonal entries which equal 1 and finally permuting back the coefficients using U. To solve the small linear systems, we used the algorithm `certSolveRedLong` from the IML library [9]. It takes a single precision dense representation of A and returns an LLL-reduced solution.

Once M is in pseudo-lower triangular form, we draw a set of relations $r_1, \ldots r_d$ which are not already rows of M, and for each r_i, $i \leq d$, we solve the system $x_i A = r_i$. We then augment M with the rows r_i for $i \leq d$ and the vectors x_i with d extra coordinates, which are all set to zero except for the i-th which is

set to -1.

$$M' := \begin{pmatrix} M \\ \cdots\cdots\cdots \\ r_i \end{pmatrix} \quad x'_i := \left(\; x_i \; \vdots \; 0 \ldots 0 \; -1 \; 0 \ldots 0 \; \right).$$

We clearly have $x'_i M' = 0$ for $i \leq d$, and the x'_i can be used to find a multiple of R_Δ as described in §2.1.

4 Numerical results

In this section, we give numerical results showing the impact of our improvements. For each timing, we specify the architecture used. All the timings were obtained with our code in C++ based on the libraries GMP [11], NTL [23], IML [9] and Linbox [19]. All timings are in CPU seconds.

4.1 Comparative timings

The state of the art concerning class group and regulator computation was established in [14], where all the timings were obtained with the SPARCStation II architecture. In addition, most of the code used at the time is unavailable now, including the HNF computation algorithm. Thus, providing a meaningful comparison between our methods and those of [14] is difficult. We chose to implement the HNF computation algorithm in a way that resembles the one of [14], but takes advantage of the libraries available today for computing the determinant and the modular HNF. We used this implementation in each different scenario. The relation collection phase is easier to compare, since our method relies on SIQS.

In the following, we will refer to the base case as the strategy consisting of finding the relation matrix without using the large prime variants or the smoothness batch test, and calculating the regulator by computing its kernel with the algorithm `nullspaceLong` from IML library. It differs from the 0 large prime case (0LP) where we use the algorithm described in §3.3 for computing the regulator, along with a relation collection phase that does not use large primes. We also denote the 1 large prime scenario by 1LP, the 2 large primes by 2LP and 2LP Batch when using batch smoothness test.

Relation collection phase In Table 1, we give the time taken to collect all necessary relations. Without large primes, we collected $|\mathcal{B}| + 100$ relations, whereas when we allow large primes we need to collect enough relations to ensure that the number of rows is larger than the number of non-empty columns. We used a 2.4 GHz Opteron with 16GB of memory and took $\Delta = 4(10^n + 3)$ with $40 \leq n \leq 70$. For each discriminant, we used the optimal parameters given in [14], including

the size of the factor base, even if we tend to reduce this parameter when optimizing the overall time. The only parameter we modified is the tolerance value for the SIQS, as a higher tolerance value is required for the large prime variations. In each case we took $B_2 = 12B_1$. It is shown in [4] that the ratio B_2/B_1 does not have an important impact on the sieving time.

Table 1. Comparative table of the relation collection time

n	0LP	1LP	2LP	2LP Batch
40	0.83	0.48	0.63	0.90
45	6.70	3.10	2.70	2.20
50	23.00	9.50	9.20	6.10
55	56.00	26.00	23.00	15.00
60	202.00	86.00	69.00	41.00
65	1195.00	513.00	354.00	227.00
70	4653.00	1906.00	1049.00	834.00

The timings in Table 1 correspond to the optimal value of the tolerance value in each case, found by trying values between 1.7 and 4, and keeping the optimum for each scenario. For 0LP, the optimal value is between 1.7 and 2.3 whereas it is around 2.3 for 1LP, 2.8 for 2LP and 3.0 for 2LP Batch. The latter case has a higher optimal tolerance value because using the batch smoothness test allows one to spend more time factoring the residues. When using Bernstein's smoothness test, we took batches of 100 residues. In our experiments, this value did not seem to have an important effect on the relation collection time. We observe in Table 1 that the use of the large prime variants has a strong impact on the relation collection phase, and that using the smoothness batch test strategy yields an additional speed-up of approximately 20% over the double large prime strategy.

Structured Gaussian elimination Structured Gaussian elimination allows us to reduce the time taken by the linear algebra phase by reducing the dimensions of the relation matrix. Our method minimizes the growth of the density and of the size of the coefficients. To illustrate the impact of the algorithm described in §3.2, we monitor in Table 2 the evolution of the dimensions of the matrix, the average Hamming weight of its rows, the extremal values of its coefficients and the time taken for computing its HNF in the case of a relation matrix corresponding to $\Delta = 4(10^{60} + 3)$. We keep track of these values after all i-way merges for some values of i between 5 and 170. The original dimensions of the matrix are 2000×1700, and the timings are obtained on a 2.4 Ghz Opteron with 32GB of memory.

In [4], the first author regularly deleted the rows having the largest coefficients. To do this, we need to create more rows than in the base case. To provide

a fair comparison between the two strategies, we used the same relation matrix resulting from a relation collection phase without large primes, and with as few rows as was required to use the same algorithm as in [14]. We therefore had to drop the regular row deletion. We also tuned the cost function to compensate for the resulting growth of the coefficients, using

$$C(r) = \sum_{1 \le |e_i| \le 8} 1 + 100 \sum_{|e_j| > 8} |e_j| \ ,$$

instead of (1).

The HNF computation consists of taking the GCD of the determinants of two different submatrices of the matrix after elimination using Linbox, and using the modular HNF of NTL with this value. Indeed, this GCD (which is likely to be relatively small) is a multiple of h_Δ. This method, combined with an elimination strategy due to Havas [13], was used in [14] and implemented in LiDIA [18]. As this implementation is no longer available, we instead refer to the timings of our code, which has the advantage of using the best linear algebra libraries available today.

Table 2. Comparative table of elimination strategies

Naive Gauss						
i	Row Nb	Col Nb	Average weight	max coeff	min coeff	HNF time
5	1189	1067	27.9	14	-17	357.9
10	921	799	49.3	22	-19	184.8
30	757	635	112.7	51	-50	106.6
50	718	596	160.1	81	-91	93.7
70	699	577	186.3	116	-104	85.6
90	684	562	205.5	137	-90	79.0
125	664	542	249.0	140	-146	73.8
160	655	533	282.4	167	-155	72.0
170	654	532	286.4	167	-155	222.4
With dedicated elimination strategy						
i	Row Nb	Col Nb	Average weight	max coeff	min coeff	HNF time
5	1200	1078	26.8	13	-12	368.0
10	928	806	42.6	20	-15	187.2
30	746	624	82.5	33	-27	100.8
50	702	580	107.6	64	-37	84.3
70	672	550	136.6	304	-676	73.4
90	656	534	157.6	1278	-1088	67.5
125	637	515	187.1	3360	-2942	63.4
160	619	497	214.6	5324	-3560	56.9
170	615	493	247.1	36761280	-22009088	192.6

Table 2 shows that the use of our elimination strategy leads to a matrix with smaller dimensions (493 rows with our method, 533 with the naive elimination) and lower density (the average weight of its rows is of 214 with our method and 282 with the naive elimination). These differences result in an improvement of the time taken by the HNF computation: 56.9 seconds with our method against 72.0 seconds with the naive Gaussian elimination. The regular cancellation of the rows having the largest coefficients over the course of the algorithm would delay the explosion of the coefficient size, but require more rows for the original matrix. This brutal increase in the size of the extremal values of the matrix can be seen in Table 2. At this point these higher values propagate during pivoting operations, and any further column elimination becomes counter-productive.

Factor base verification The improvements in the relation collection phase have an impact on the factor base verification. The impact of the smoothness batch test is straightforward, whereas the large prime variants act in a more subtle way. Indeed, we create many more relations when using the large prime variants, and the relations created involve primes of larger norm. Therefore, a given prime not in $\mathcal{B}$ of norm less than $6 \log^2 \Delta$ is more likely to appear in a relation, and thus not to need to be verified. Table 3 shows the impact of the large prime variants and of the batch smoothness test on the verification time. We used a 2.4 GHz Opteron with 16GB of memory. We considered discriminants of the form $\Delta = 4(10^n + 3)$ for n between 40 and 70, and we chose in every case the factor base giving the best results for the base scenario.

Table 3. Comparative table of the factor base verification time

n	0LP	1LP	2LP	2LP Batch
40	17.0	11.0	11.0	6.2
45	77.0	44.0	30.0	18.0
50	147.0	85.0	52.0	43.0
55	308.0	167.0	134.0	110.0
60	826.0	225.0	282.0	274.0
65	8176.0	1606.0	1760.0	1689.0
70	9639.0	4133.0	5777.0	2706.0

Regulator computation Our method for computing the regulator avoids computing the relation matrix kernel. Instead, we need to solve a few linear systems involving the matrix resulting from the Gaussian elimination. To illustrate the impact of this algorithm, we used the relation matrix obtained in the base case for discriminants of the form $4(10^n + 3)$ for n between 40 and 70. The timings are obtained on a 2.4GHz Opteron with 16GB of memory.

Table 4. Comparative table of regulator computation time

n	Kernel Computation	System Solving
40	15.0	6.2
45	18.0	8.3
50	38.0	20.0
55	257.0	49.0
60	286.0	103.0
65	5009.0	336.0
70	10030.0	643.0

In Table 4, the timings corresponding to our system solving approach are taken with seven kernel vectors. However, in most cases only two or three vectors are required to compute the regulator. As most of the time taken by our approach is spent on system solving, we see that computing fewer kernel vectors would result in an improvement of the timings, at the risk of obtaining a multiple of the regulator.

Overall time We have studied the individual impact of our improvements on each stage of the algorithm. We now present their effect on the overall time taken by the algorithm, including the factor base verification time, for discriminants of the form $\Delta = 4(10^n + 3)$ with $40 \leq n \leq 70$ on a 2.4 GHz Opteron with 16GB of memory. We used the same parameters as in [14], except for the tolerance and the size of the factor base. We notice in Table 5 that the optimal size of the factor base is smaller when we use improvements for the sieving phase. For example the optimal size for the double large prime variant is half the one of the base case scenario. This results in an improvement in the HNF and regulator computation whereas the relation collection time can remain unchanged, or even increase. The tolerance value we chose varies only with the strategy, but not with the size of the discriminant. We chose 2.0 for the base case and 0LP whereas we set it to 2.3 for 1LP, 2.8 for 2LP and 3.0 for 2LP Batch. We eliminated columns of weight up to $w = 150$ since Table 2 indicates that further elimination is counter-productive.

Table 5 shows that there is an overall speed-up of of a factor of 2 for the smallest discriminants and 4 for the largest. The base case with the largest discriminants suffers from the necessity of finding some relations in a more randomized way. This ensures that we can get full rank submatrices of the relation matrix after the Gaussian elimination to compute a small multiple of h_Δ. Matrices produced using the large prime variants do not need this extra step, even with the largest discriminants. This naturally affects the sieving time, since we cannot use SIQS for that purpose, but also affects phases relying on linear algebra. Indeed, elimination produces a matrix with larger entries and dimensions.

Table 5. Effect on the overall time

n	strategy	$\lvert\mathcal{B}\rvert$	relations	elimination	HNF	regulator	verification	total
40	base	400	0.8	0.1	3.2	14.6	16.8	35.6
	0LP	400	0.7	0.1	2.2	6.0	16.6	25.7
	1LP	300	0.8	0.2	2.5	6.4	13.1	23.1
	2LP	250	1.7	0.3	4.8	8.7	18.0	33.3
	2LP Batch	250	0.5	0.2	3.6	6.7	4.4	15.5
45	base	500	6.7	0.1	5.1	18.0	77.0	107.0
	0LP	500	5.9	0.2	4.9	10.0	85.0	106.0
	1LP	400	4.0	0.4	6.0	11.0	50.0	71.0
	2LP	350	3.8	0.5	12.0	17.0	36.0	69.0
	2LP Batch	350	2.6	1.1	9.0	14.0	30.0	57.0
50	base	750	23.0	0.3	16.0	38.0	147.0	224.0
	0LP	700	21.0	0.4	15.0	20.0	147.0	203.0
	1LP	450	20.0	0.4	10.0	17.0	108.0	155.0
	2LP	400	14.0	0.8	22.0	23.0	74.0	133.0
	2LP Batch	400	10.0	0.6	21.0	25.0	62.0	119.0
55	base	1200	129.0	1.9	60.0	257.0	308.0	756.0
	0LP	1300	47.0	0.7	52.0	49.0	265.0	414.0
	1LP	650	61.0	0.7	28.0	33.0	255.0	378.0
	2LP	550	40.0	1.1	48.0	48.0	177.0	313.0
	2LP Batch	550	34.0	1.0	47.0	48.0	141.0	271.0
60	base	1700	322.0	2.9	95.0	286.0	830.0	1535.0
	0LP	1700	187.0	1.3	106.0	103.0	846.0	1244.0
	1LP	750	309.0	1.0	45.0	64.0	865.0	1284.0
	2LP	700	143.0	2.1	152.0	137.0	365.0	799.0
	2LP Batch	700	142.0	1.8	103.0	100.0	309.0	655.0
65	base	2700	10757.0	12.0	652.0	5009.0	8176.0	24607.0
	0LP	2700	1225.0	2.8	489.0	336.0	3676.0	5730.0
	1LP	1900	1003.0	15.0	318.0	262.0	2984.0	4583.0
	2LP	1200	753.0	4.7	525.0	398.0	1943.0	3624.0
	2LP Batch	1000	1030.0	35.0	199.0	219.0	1642.0	3125.0
70	base	3700	17255.0	24.0	1869.0	10031.0	9639.0	38818.0
	0LP	3600	4934.0	19.0	1028.0	644.0	9967.0	16591.0
	1LP	2500	3066.0	17.0	845.0	646.0	9005.0	13579.0
	2LP	1700	2414.0	27.0	2054.0	1295.0	4590.0	10379.0
	2LP Batch	1700	2588.0	20.0	1372.0	934.0	5078.0	9991.0

4.2 Large example

The improvements we described allow us to compute class groups and regulators of real number fields with larger discriminants than was previously possible. The key is to parallelize the relation collection and verification phase, while the linear algebra has to be performed the usual way. These methods were successfully used in [4] to compute the class group structure of an imaginary quadratic field with a 110-digit discriminant. We used a cluster with 260 2.4GHz Xeon cores to

compute a relation matrix corresponding to the discriminant $\Delta_{110} := 4(10^{110}+3)$ in 4 days. We allowed two large primes, used a tolerance value of 3.0, tested batches of 100 residues, took $w = 250$ and set $|\mathcal{B}| = 13000$. Then, we used three 2.4 GHz Opterons with 32GB of memory each to compute determinants of full-rank submatrices of the relation matrix after the Gaussian elimination in 1 day, and one 2.4GHz Opteron to compute the HNF modulo the GCD of these determinants in 3 days. We had to find 4018 extra relations during the verification phase that took 4 days on 96 2.4GHz Xeon cores. We thus obtained that

$$Cl_{\Delta_{110}} \cong \mathbb{Z}/12\mathbb{Z} \times \mathbb{Z}/2\mathbb{Z} \ , \tag{2}$$

and the corresponding regulator is

$$R_{\Delta_{110}} \approx 70795074091059722608293227655184666748799878533480399.6730200233 \ .$$

We estimate that it would take two weeks (4000 relations per day) to complete the relation collection for Δ_{120} with the same factor base as Δ_{110}, thus requiring a similar time for the linear algebra.

5 Conclusions

Recently, our work has been extended to the problems of principal ideal testing and solving the discrete logarithm problem in the ideal class group [5]. The double large prime variant and improvements to relation generation translated directly to improvements in this context. However, HNF computations are not required for this problem, and linear system solving over $\mathbb{Z}$ can be used instead. The numerical results were used to give estimates for discriminant sizes that offer equivalent security to recommended sizes of RSA moduli.

Some possibilities for further improvements remain to be investigated. For example, a lattice sieving strategy could be used to sieve $\varphi(x, y)$ instead of $\varphi(x, 1)$. Factor refinement and coprime factorization techniques may be a useful alternative to Bernstein's batch smoothness test. Multiple large primes have been successfully used for integer factorization and could also be tried in our context.

There is also still room for improvement to the linear algebra components. For example, a HNF algorithm that exploits the natural sparseness of the relation matrix, perhaps as a black-box algorithm, would be useful. If such an algorithm were available, we could reconsider using Gaussian elimination techniques since they induce a densification of the matrix. We could also study the effect of other dense HNF algorithms in existing linear algebra packages such as KASH, Pari, Sage and especially MAGMA which seems to have the most efficient HNF algorithm for our types of matrices. In that case, we would need the elimination phase regardless of how these algorithms are affected by the density and the size of the coefficients of the matrix. Indeed, we cannot afford manipulating a dense representation of the matrix before the Gaussian elimination phase.

References

1. E. Bach, *Explicit bounds for primality testing and related problems*, Math. Comp. **55** (1990), no. 191, 355–380.
2. ———, *Improved approximations for Euler products*, Number Theory: CMS Proc., vol. 15, Amer. Math. Soc., Providence, RI, 1995, pp. 13–28.
3. D. Bernstein, *How to find smooth parts of integers*, submited to *Mathematics of Computation*.
4. J-F. Biasse, *Improvements in the computation of ideal class groups of imaginary quadratic number fields*, to appear in *Advances in Mathematics of Communications*, 2010.
5. J-F. Biasse, M. J. Jacobson, Jr., A. K. Silvester, *Security estimates for quadratic field based cryptosystems*, to appear in ACISP 2010.
6. J. Buchmann, *A subexponential algorithm for the determination of class groups and regulators of algebraic number fields*, Séminaire de Théorie des Nombres (Paris), 1988-89, pp. 27–41.
7. J. Buchmann and S. Düllmann, *Distributed class group computation*, Festschrift aus Anlaß des sechzigsten Geburtstages von Herrn Prof. Dr. G. Hotz, Universität des Saarlandes, 1991, and Teubner, Stuttgart, 1992, pp. 69–79.
8. S. Cavallar, *Strategies in filtering in the number field sieve*, ANTS-IV: Proceedings of the 4th International Symposium on Algorithmic Number Theory, Lecture Note in Computer Science, vol. 1838, Springer-Verlag, 2000, pp. 209–232.
9. Z. Chen, A. Storjohann, and C. Fletcher, *IML: Integer Matrix Library*, Software, 2010, see http://www.cs.uwaterloo.ca/ astorjoh/iml.html.
10. B. Dodson, P. C. Leyland, A. K. Lenstra, A. Muffett, and S. Wagstaff, *MPQS with three large primes*, ANTS-V: Proceedings of the 5th International Symposium on Algorithmic Number Theory, Lecture Note in Computer Science, vol. 2369, Springer-Verlag, 2002, pp. 446–460.
11. GMP, *The GNU multiple precision bignum library*, Software, 2010, see http://gmp-lib.org/.
12. J. E. Gower and S. Wagstaff, *Square form factorization*, Mathematics of Computation **77** (2008), 551–588.
13. G. Havas and B.S. Majewski, *Integer matrix diagonalization*, Journal of Symbolic Computing **24** (1997), 399–408.
14. M. J. Jacobson, Jr., *Subexponential class group computation in quadratic orders*, Ph.D. thesis, Technische Universitt Darmstadt, Darmstadt, Germany, 1999.
15. M. J. Jacobson, Jr., R. Scheidler, and H. C. Williams, *The efficiency and security of a real quadratic field based key exchange protocol*, Public-Key Cryptography and Computational Number Theory (Warsaw, Poland), de Gruyter, 2001, pp. 89–112.
16. M. J. Jacobson, Jr. and H. C. Williams, *Solving the Pell equation*, CMS Books in Mathematics, Springer-Verlag, 2009, ISBN 978-0-387-84922-5.
17. A. K. Lenstra and M. S. Manasse, *Factoring with two large primes (extended abstract)*, Advances in Cryptology - EUROCRYPT '90, Lecture Note in Computer Science, vol. 473, Springer-Verlag, 1991, pp. 72–82.
18. LiDIA Group, *LiDIA: a* `c++` *library for computational number theory*, Software, Technische Universität Darmstadt, Germany, 1997, see http://www.informatik.tu-darmstadt.de/TI/LiDIA.
19. LinBox, *Project LinBox: Exact computational linear algebra*, Software, 2010, see http://www.linalg.org/.

20. S. Louboutin, *Computation of class numbers of quadratic number fields*, Math. Comp. **71** (2002), no. 240, 1735–1743.
21. M. Maurer, *Regulator approximation and fundamental unit computation for real quadratic orders*, Ph.D. thesis, Technische Universitt Darmstadt, Darmstadt, Germany, 1999.
22. J. Milan, *Tifa*, Software, 2010, http://www.lix.polytechnique.fr/Labo/Jerome.-Milan/tifa/tifa.xhtml.
23. V. Shoup, *NTL: A Library for doing Number Theory*, Software, 2010, http://www.-shoup.net/ntl.
24. U. Vollmer, *An accelerated Buchmann algorithm for regulator computation in real quadratic fields*, Algorithmic Number Theory — ANTS-V, Lecture Notes in Computer Science, vol. 2369, 2002, pp. 148–162.