Improvements in the computation of ideal class groups of imaginary quadratic number fields

Jean-François Biasse

Advances in Mathematics of Communications
Volume X, No. 0X, 200X, X–XX

Web site: http://www.aimSciences.org

IMPROVEMENTS IN THE COMPUTATION OF IDEAL CLASS GROUPS OF IMAGINARY QUADRATIC NUMBER FIELDS

Jean-François Biasse

LIX - École Polytechnique
91128 Palaiseau , France

(Communicated by Renate Scheidler)

Abstract. We investigate improvements to the algorithm for the computation of ideal class group described by Jacobson in the imaginary quadratic case. These improvements rely on the large prime strategy and a new method for performing the linear algebra phase. We achieve a significant speed-up and are able to compute 110-decimal digits discriminant ideal class group in less than a week.

1. Introduction

Given a fundamental discriminant Δ, it is known that the corresponding ideal class group $\mathrm{Cl}(\Delta)$ of the order $\mathcal{O}_\Delta$ of discriminant Δ in $\mathbb{K} = \mathbb{Q}(\sqrt{\Delta})$ is a finite abelian group that can be decomposed as

$$\mathrm{Cl}(\Delta) \simeq \bigoplus_i \mathbb{Z}/d_i\mathbb{Z},$$

where the divisibility condition $d_i | d_{i+1}$ holds. In this paper we investigate improvements in the computation of the group structure of $\mathrm{Cl}(\Delta)$: that is determining the d_i, which is of both cryptographic and number theoretic interest. Indeed some key-exchange protocols relying on the difficulty of solving the discrete logarithm problem (DLP) in imaginary quadratic orders have been proposed [3, 9] and solving instances of the DLP is closely related to finding the group structure of $\mathrm{Cl}(\Delta)$.

In 1968 Shanks [18] proposed an algorithm relying on the baby-step giant-step method in order to compute the structure of the ideal class group of an imaginary quadratic number field in time $O\left(|\Delta|^{1/4+\epsilon}\right)$, or $O\left(|\Delta|^{1/5+\epsilon}\right)$ under the extended Riemann hypothesis [13]. This allows us to compute class groups of discriminants having up to 20 or 25 decimal digits. Then a subexponential strategy was described in 1989 by Hafner and McCurley [8]. The expected running time of this method is

$$e^{\left(\sqrt{2}+o(1)\right)\sqrt{\log\Delta\log\log\Delta}}.$$

Buchman and Düllmann [2] computed class groups with discriminants of around 50 decimal digits using an implementation of this algorithm. An improvement of this method was published by Jacobson in 1999 [10]. He achieved a significant speed-up by using sieving strategies to generate the matrix of relations. He was able to compute the structure of class groups of discriminants having up to 90

2000 *Mathematics Subject Classification:* Primary: 58F15, 58F17; Secondary: 53C35.

Key words and phrases: Ideal Class group, index calculus, large prime variant, Gaussian elimination, Hermite normal form.

The author is supported by a DGA grant.

decimal digits. More recently Sutherland [21] used generic methods in order to compute class groups with discriminants having 100 decimal digits. Unlike the previous algorithms, this one relies heavily on the particular structure of $\mathrm{Cl}(\Delta)$ thus obtaining variable performances depending on the values of Δ.

Our approach is based on that of Jacobson, using new techniques to accelerate both the sieving phase and the linear algebra phase; we have obtained the group structure of class groups of 110 decimal digit discriminants.

2. The ideal class group

In this section we give essential results concerning the ideal class group and the subexponential strategies for computing its structure. For a more detailed description of the theory of ideal class groups we refer to [5] and [16]. In the following, Δ denotes a fundamental discriminant and $\mathbb{K} = \mathbb{Q}(\sqrt{\Delta})$ is the corresponding number field.

2.1. Description. Elements of $\mathrm{Cl}(\Delta)$ are obtained from fractional ideals of the ring of integer $\mathcal{O}_{\mathbb{K}}$ of $\mathbb{K}$. We denote by $\mathcal{I}$ the set of fractional ideals, which are $\mathbb{Z}$-modules of the form:

$$\mathfrak{a} = q\left(a\mathbb{Z} + \frac{b+\sqrt{\Delta}}{2}\mathbb{Z}\right)$$

where a and b are integers with $b \equiv \Delta \bmod 2$ and q is a rational number. The prime ideals are the $\mathfrak{p} \in \mathcal{I}$ for which there exists a prime number p such that:

$$\mathfrak{p} = p\mathbb{Z} + \frac{b_p+\sqrt{\Delta}}{2}\mathbb{Z} \quad \text{or} \quad \mathfrak{p} = p\mathbb{Z} \text{ (case inert)}$$

For every $\mathfrak{a} \in \mathcal{I}$ there exist uniquely determined prime ideals $\mathfrak{p}_1, \ldots, \mathfrak{p}_n$ and exponents $e_1, \ldots, e_n$ in $\mathbb{Z}$ such that

$$\mathfrak{a} = \mathfrak{p}_1^{e_1} \ldots \mathfrak{p}_n^{e_n}.$$

Definition 2.1 (Ideal Class group)**.** Let $\mathcal{I}$ be the set of fractional ideals of $\mathbb{K}$ and $\mathcal{P} = \{(\alpha) \in \mathcal{I}, \alpha \in \mathbb{K}\}$ the subset of principal ideals. We define the ideal class group of Δ as :

$$\mathrm{Cl}(\Delta) := \mathcal{I}/\mathcal{P}$$

Unlike $\mathcal{I}$, the ideal class group $\mathrm{Cl}(\Delta)$ is a finite group. Its order is called the class number and usually denoted by $h(\Delta)$. It grows like $|\Delta|^{1/2+\epsilon}$ as it is shown in [19].

2.2. Computing the group structure. The algorithm for computating the group structure of $\mathrm{Cl}(\Delta)$ is divided into two major phases: relation collection and linear algebra. In the first phase, we begin by precomputing a factor base $\mathcal{B} = \{\mathfrak{p}_1, \ldots, \mathfrak{p}_n\}$ of non-inert prime ideals satisfying $\mathcal{N}(\mathfrak{p}_i) \leq B$, where B is a smoothness bound. Then we look for relations of the form

$$(\alpha) = \mathfrak{p}_1^{e_1} \ldots \mathfrak{p}_n^{e_n},$$

where $\alpha \in \mathbb{K}$. Every n-tuple $[e_1, \ldots, e_n]$ collected becomes a row of what we will refer to as the relation matrix $A \in \mathbb{Z}^{m \times n}$. We have from [1] the following important result:

Theorem 2.2. *Let Λ be the lattice spanned by the set of the possible relations. Assuming GRH, if $B \geq 6\log^2 \Delta$, then we have*

$$\mathrm{Cl}(\Delta) \simeq \mathbb{Z}^n/\Lambda.$$

After the relation collection phase we can test if A has full rank and if its rows generate Λ using methods described in §3.3. If it is not the case then we have to compute more relations. From now on we assume that A has full rank and that its rows generate Λ.

The linear algebra phase consists of computing the Smith Normal Form (SNF) of A. Any matrix A in $\mathbb{Z}^{n\times n}$ with non zero determinant can be written as

$$A = V^{-1}\begin{pmatrix} d_1 & 0 & \dots & 0 \\ 0 & d_2 & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & d_n \end{pmatrix} U^{-1}$$

where $d_{i+1}|d_i$ for all $1 \leq i < n$ and U and V are unimodular matrices in $\mathbb{Z}^{n\times n}$. The matrix $\mathrm{diag}(d_1, \dots, d_n)$ is called the SNF of A. If $m = n$ and $\mathrm{diag}(d_1, \dots, d_n) = \mathrm{SNF}(A)$ then

$$\mathrm{Cl}(\Delta) \simeq \bigoplus_{i=1}^{n} \mathbb{Z}/d_i\mathbb{Z}.$$

This reduces the problem of computing the group structure of $\mathrm{Cl}(\Delta)$ to computing the SNF of a relation matrix A in $\mathbb{Z}^{n\times n}$. For an arbitrary A in $\mathbb{Z}^{m\times n}$ we start by computing the Hermite Normal Form (HNF) of A. A matrix H is said to be in HNF if it has the shape

$$H = \begin{pmatrix} h_{1,1} & 0 & \dots & 0 \\ \vdots & h_{2,2} & \ddots & \vdots \\ \vdots & \vdots & \ddots & 0 \\ * & * & \dots & h_{n,n} \\ \hdashline (0) & & & \end{pmatrix}$$

where $0 \leq h_{ij} < h_{ii}$ for all $j < i$ and $h_{ij} = 0$ for all $j > i$. For each matrix A in $\mathbb{Z}^{m\times n}$ there exists an H in HNF and a unimodular matrix W in $\mathbb{Z}^{m\times m}$ such that

$$H = WA.$$

The upper block of H is a $n \times n$ relation matrix whose SNF provides us the group structure of $\mathrm{Cl}(\Delta)$. There is an index l such that $h_{i,i} = 1$ for every $i \geq l$. The upper left $l \times l$ submatrix of H is called the essential part of H. In order to compute the group structure of $\mathrm{Cl}(\Delta)$ it suffices to compute the SNF of the essential part of H, which happens to have small dimension in our context.

2.3. THE USE OF SIEVING FOR COMPUTING THE RELATION MATRIX. The use of sieving to create the relation matrix was first described by Jacobson [10]. Here we follow the approach of [11], which relies on the following lemma:

Lemma 2.3. *If* $\mathfrak{a} = \left(a\mathbb{Z} + \frac{b+\sqrt{\Delta}}{2}\mathbb{Z}\right)$ *with* $a > 0$*, then for all* x, y *in* $\mathbb{Z}$ *there exists a* $\mathfrak{b}$ *in* $\mathcal{I}$ *such that* $\mathfrak{ab} \in \mathcal{P}$ *and*

$$\mathcal{N}(\mathfrak{b}) = ax^2 + bxy + \frac{b^2 - \Delta}{4a}y^2.$$

The strategy for finding relations is the following: We start with $\mathfrak{a} = \prod_i \mathfrak{p}_i^{e_i}$ which is $\mathcal{B}$-smooth. Then we choose a sieve radius R satisfying $R \approx \sqrt{|\Delta|/2}/\mathcal{N}(\mathfrak{a})$ and we look for values of $x \in [-R, R]$ such that $\varphi(x, 1)$ is B-smooth where

$$\varphi(x, y) = ax^2 + bxy + \frac{b^2 - \Delta}{4a}y^2,$$

which allows us to find $\mathfrak{b} = \prod_i \mathfrak{p}_i^{f_i}$ satisfying $\mathfrak{ab} = (\gamma)$ for some γ in $\mathbb{K}$. The $\mathfrak{p}_i$ and f_i are deduced from the decomposition $\varphi(x, 1) = \prod_i p_i^{v_i}$. For more details we refer to [11]. This method yields the relation

$$(\gamma) = \prod_i \mathfrak{p}_i^{e_i + f_i}.$$

Now given a binary quadratic form $\varphi(x, y) = ax^2 + bxy + cy^2$ of discriminant Δ, we are interested in finding values of $x \in [-R, R]$ such that $\varphi(x, 1)$ is B-smooth. This can be done trivially by testing all the possible values of x, but there is a well-known method for pre-selecting some values of x in $[-R, R]$ that are going to be tested, namely the quadratic sieve (introduced by Pomerance [17]). It consists in initializing to 0 an array S of length $2R + 1$ and precomputing the roots r_i' and r_i'', or the double root r_i', of $\varphi(x, 1) \mod p_i$ for each $p_i \leq B$ such that $\left(\frac{\Delta}{p_i}\right) \neq -1$. Then for each x in $[-R, R]$ of the form $x = r_i + kp_i$ for some k we add $\lfloor \log p_i \rfloor$ to $S[x]$. At the end of this procedure, if $\varphi(x, 1)$ is B-smooth, then $S[x] \approx \log \varphi(x, 1)$. As $\varphi(x, 1) \approx \sqrt{\Delta/2}R$, we set a bound

$$F = \log\left(\sqrt{\frac{\Delta}{2}}R\right) - T\log(p_n) \tag{1}$$

where T is a number representing the tolerance to rounding errors due to integer approximations. We then perform a trial division test on every $\varphi(x, 1)$ such that $S[x] \geq F$.

3. PRACTICAL IMPROVEMENTS

In this section we describe the improvements that allowed us to achieve a significant speed-up with respect to the existing algorithm and the computation of class group structures of large discriminants. Our contribution is to take advantage of the large prime variants, of an algorithm due to Vollmer [22] for the SNF which had not been implemented in the past, and of special Gaussian elimination techniques.

3.1. 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. Jacobson considered analogous variants for class group computation [10], but the speed-up of the relation collection phase was achieved at the price of such a slow-down of the linear algebra that it did not

significantly improve the overall time. The main idea is the following: We define the "small primes" to be the prime ideals in the factor base and the small prime bound as the corresponding bound $B_1 = B$. Then we define a large prime bound B_2. During the relation collection phase we choose not to restrict ourselves to relations only involving primes $\mathfrak{p}$ in $\mathcal{B}$ but we also keep relations of the form

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

for $\mathfrak{p}_i$ in $\mathcal{B}$, and for $\mathfrak{p}, \mathfrak{p}'$ of norm less than B_2. We will respectively refer to them as 1-partial relations and 2-partial relations. Keeping partial relations only involving one large prime is the single large prime variant, whereas keeping two of them is the double large prime variant which was first described by Lenstra and Manasse [12]. In this paper we do not consider the case of more large primes, but it is a possibility that has been studied in the context of factorization [14].

Partial relations may be identified as follows. Let m be the residue 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. $m \leq B_2$;
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 $p' = \mathcal{N}(\mathfrak{p}')$. After testing if we are in Cases 1, 2, or 3 we have to factorize the residue. We have done that using Milan's implementation of the SQUFOF algorithm [15] based on the theoretical work of [7].

Even though we might have to factor the residue, collecting a partial relation is much faster than collecting a full relation because the probability that $\mathcal{N}(\mathfrak{b})$ is B_2-smooth is much greater than the probability that it is B_1-smooth. This improvement in the speed of the relation collection phase comes at a price: The number of columns in the relation matrix is much greater, thus preventing us from running the linear algebra phase directly on the resulting relation matrix and forcing us to find many more relations since since we have to produce a full rank matrix. We will see in §3.2 how we reduced the dimensions of the relation matrix using Gaussian elimination techniques and in §4 how to optimize the parameters to make the creation of the relation matrix faster, even though there are many more relations to be found.

3.2. Gaussian elimination techniques. 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 [12] 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 find recombinations of partial relations into full relations.

In this paper we rather follow the approach of Cavallar [4], developed for the number field sieve, which uses Gaussian elimination on columns without distinguishing those corresponding to the large primes from the others. One of the main differences between our relation matrices and the matrices produced in the number field sieve is that our entries are in $\mathbb{Z}$ rather than $\mathbb{F}_2$, thus obliging us to monitor the evolution of the size of the coefficients. Indeed, eliminating columns at the price of

an explosion of the size of the coefficients can be counter-productive in preparation for the HNF algorithm.

In what follows we will use a few standard definitions that we briefly recall here. First, subtracting two rows is called *merging*. This is because rows are stored as lists of the non-zero entries sorted with respect to the corresponding columns and subtracting them corresponds to merging the two sorted lists. If two rows r_1 and r_2 share the same prime $\mathfrak{p}$ with coefficients c_1 and c_2 respectively then multipling 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 dimension 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. Unfortunately, the density of the rows and the size of the coefficients increase during the course of the algorithm, thus obliging us to use 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.

First we have to define a cost function defined over the set of the rows encapsulating the difficulty induced for the HNF algorithm. In factorization, we want to find a vector in the kernel of the relation matrix which is defined over $\mathbb{F}_2$; the only property of the row that really matters is its Hamming weight. In our context, we need to minimize the Hamming weight of the row, but we also have to take into account the size of the coefficients. Different cost functions lead to different elimination strategies. Our cost function was determined empirically: We took the number of non-zero entries, counting c times those whose absolute value was above a bound Q, where c is a positive number. If $r = [e_1, \ldots, e_n]$ corresponds to $(\alpha) = \prod_i \mathfrak{p}_i^{e_i}$ then

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

Indeed as we will see matrices with small entries are better suited for the HNF algorithm described in §3.3. Let us assume now that we are to perform a k-way merge on a given column. We construct a complete graph $\mathcal{G}$ of size k as follows:

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

Finding the best sequence of pivots with respect to the cost function c we chose 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 $\mathcal{T}$.

Unfortunately, some coefficients might grow during the course of column eliminations despite the use of this strategy. Once a big coefficient is created in a given row r, it is likely to spread to other rows once r is involved in another column elimination. We must therefore discard such rows as quickly as possible. In our implementation we chose to do it regularly: Once we performed all the k-way merges for $k \leq 10 \cdot i$ and $i = 1, \ldots, w/10$ we discard a fixed number K of the rows containing the largest coefficients.

We show in Table 1 the effect of the use of a cost function taking into account the size of the coefficients and the regular discard of the worst rows for $\Delta = -4(10^{70}+1)$ with $c = 100$, $Q = 8$ and $K = 10$. We kept track of the evolution of the dimensions of the matrix, the average Hamming weight of the rows and the maximum and minimum size of the coefficients. In the first case we use the traditional cost function

that only takes into account the Hamming weight of the rows and we keep deleting the worst rows regularly; this corresponds to taking $c = 1$ and $K = 10$. In the second case, we use the cost function described above but without row elimination by setting $c = 100$ and $K = 0$. In the third case, we combine the two ($c = 100$ and $K = 10$). We clearly see that the coefficients are properly monitored only in the latter case. Indeed using a cost function that does not take into account the size of the coefficients and just discarding the worst rows regularly seems more efficient in terms of reduction of the matrix dimension, but the row corresponding to $i = 12$ (that is to say after all the 120-way merges) clearly shows that we run the risk of an explosion of the coefficients.

Figure 1. Comparative table of elimination strategies

Without score depending on the size of the coefficients					
i	Row Nb	Col Nb	Average weight	max	min
0	38752	45975	22	10	-10
2	2334	1668	76	21	-20
4	2123	1477	117	52	-56
6	2028	1402	146	59	-62
8	1951	1345	175	72	-65
10	1890	1304	203	193	-196
12	1836	1270	219	212	-2147483648
Without row elimination					
i	Row Nb	Col Nb	Average weight	max	min
0	38752	45975	22	10	-10
2	2373	1687	79	30	-40
4	2224	1538	118	67	-50
6	2158	1472	148	71	-132
8	2117	1431	179	2648	-10568
10	2097	1411	196	347136	-337920
12	2080	1394	214	268763136	-173162496
With adapted score and row elimination					
i	Row Nb	Col Nb	Average weight	max	min
0	38752	45975	22	10	-10
2	2357	1691	76	17	-17
4	2176	1530	114	27	-30
6	2074	1448	149	37	-37
8	2013	1407	177	43	-43
10	1958	1372	199	44	-45
12	1908	1342	224	54	-53

3.3. Vollmer's algorithm for computing the HNF. In [10] it has been observed that the algorithm used to compute the HNF of the relation matrix relied heavily on the sparsity of the matrix. While recombinations of the kind described in [12] or the techniques of §3.2 reduce the dimensions of the matrix, they also dramatically increase the density of the matrix, thus slowing down the computation of the HNF. We had to find an HNF algorithm whose features were adapated to our situation. Vollmer described in [22] an algorithm of polynomial complexity depending on the capacity to solve diophantine linear systems, but not on the density of the matrix. It was not implemented at the time because there was no efficient

diophantine linear system solver available. We implemented Vollmer's algorithm using the IML [20] library provided by Storjohann.

Here we give a brief description of the algorithm (for more details we refer to [22]). We assume we have created an $m \times n$ relation matrix A of full rank. For each $i \leq n$, we define two matrices

$$A_i = \begin{pmatrix} a_{m,1} & \dots & a_{1,1} \\ \vdots & & \vdots \\ a_{m,i} & \dots & a_{i,1} \end{pmatrix} \quad \text{and} \quad e_i = \begin{pmatrix} 0 \\ \vdots \\ 0 \\ 1 \end{pmatrix}.$$

For each i, we define h_i to be the minimal denominator of a rational solution of the system

$$A_i x = e_i;$$

this is computed using the function `MinCertifiedSol` of IML. In [22] it is shown that

$$h(\Delta) = \prod_i h_i.$$

Fortunately, analytic formulae allow us to compute a bound h_* such that

$$h_* \leq h(\Delta) < 2h_*,$$

so we do not have to compute h_i for every $i \in [1, n]$. In addition, the matrices produced for the computation of the group structure of $\mathrm{Cl}(\Delta)$ have small essential part, which keeps the number of diophantine systems to solve small (about the same size as the number of columns of the essential part) as shown in [22].

Algorithm 1 Computation of the class number

Input: Δ, relation matrix A of full rank and h_*
Output: $h(\Delta)$

```
h ← 1
i ← 1
while h < h_* do
    Compute the minimal denominator h_i of a solution of A_i · x = e_i
    h ← h · h_i
    i ← i + 1
end while
return h
```

We can compute the essential part of the HNF of A with a little extra effort involving only modular reductions of coefficients; we refer to [22] for more details. This part of the algorithm is highly dependent on the performance of the diophantine solver we use, which in turn is mostly influenced by the number of columns of the matrix and the size of the coefficients. The bechmarks available [20] show that the algorithm runs much faster on matrices with 3-bit coefficients, which is why we took coefficient size into account in the cost function for the Gaussian elimination.

4. Optimization of the parameters

In this section we proceed to optimize the parameters involved in the relation collection phase. Each parameter has an effect on the overall time taken to compute the group structure of $\mathrm{Cl}(\Delta)$. Recall (1) giving the bound F; when we collect partial relations it should be adapted in the following way:

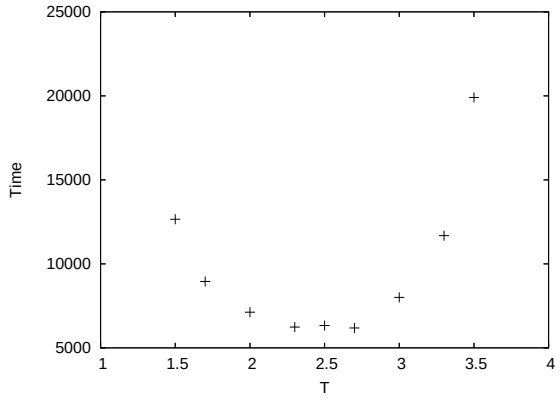
$$F = \log\left(\sqrt{\frac{\Delta}{2}}R\right) - T\log B_2,$$

where B_2 is the large prime bound.

4.1. Optimization of T.

The parameter T represents the tolerance to rounding errors in the traditional sieving algorithms. Its value is empirically determined, and usually lies in the interval $[1, 2]$. In the large prime variant it also encapsulates the number of large primes we want to allow. Indeed, if there were no rounding errors one would expect this value to be 1 for one large prime and 2 for two large primes. In practice, we can exhibit an optimum value which differs slightly from what we would expect. In figure 2 we show the overall running time of the algorithm when the parameter T varies between 1.5 and 3.5 for the discriminant $\Delta = -4(10^{75}+1)$. The size of the factor base taken is 3250, the ratio B_2/B_1 equals 120, and we allow two large primes.

Figure 2. Optimum value of T



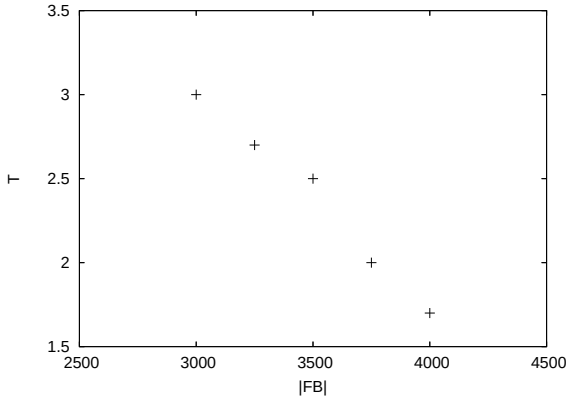
One of the main issues for determining the optimal value of T is that it tends to shift when one modifies the value of B_1, the rest being unchanged. Indeed, if for example $B_2/B_1 = 120$ then

$$F = \log\left(\sqrt{\frac{\Delta}{2}}R\right) - T\log 120B_1,$$

so when we increase B_1 we have to lower T to compensate. Figure 3 illustrates this phenomenon on the example $\Delta = -4(10^{75}+1)$, with two large primes.

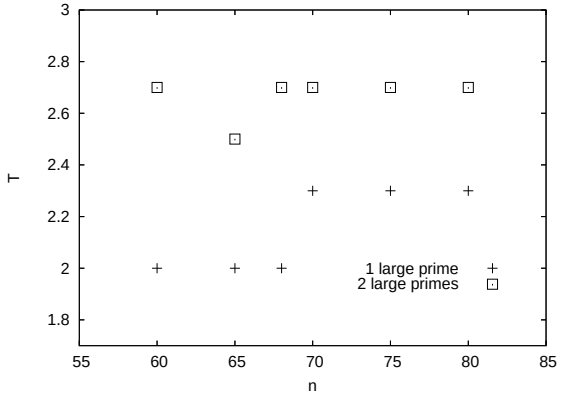
In Figure 4 we study the evolution of the optimal value of T for the single and double large prime variants on discriminants of the form $-4(10^n+1)$ where n ranges between 60 and 80. It appears that, as we expected, the optimal value for the double

FIGURE 3. Effect of $|FB|$ on the optimal value of T



large prime variant is greater than the one corresponding to the single large prime variant. This value is between 2 and 2.3 for one large prime and around 2.7 when we allow two large primes.

FIGURE 4. Optimal value of T when n varies



4.2. THE SIZE OF THE FACTOR BASE. The optimal size of the factor base reflects the trade-of between the time spent on the relation collection phase and on the linear algebra phase. This optimum is usually not the size that minimizes the time spent on the relation collection phase. To illustrate this, Figure 5 shows the time taken by the algorithm for $\Delta = -4(10^{75} + 1)$ with $B_2/B_1 = 120$ and the corresponding optimal T.

The optimal size of the factor base increases with the size of the discriminant. Figure 6 shows the optimal size of the factor base for discriminants of the form $-4(10^n + 1)$ as n ranges between 60 and 80 for both one large prime and two large primes. We notice that the single large prime variant requires smaller factor bases than without large primes, and bigger factor bases than the double large prime variant.

FIGURE 5. Optimal value of $|FB|$

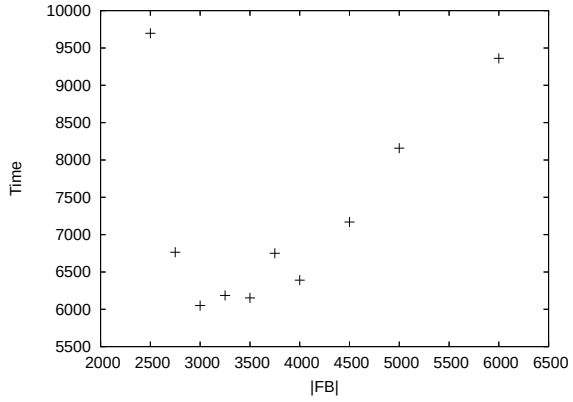
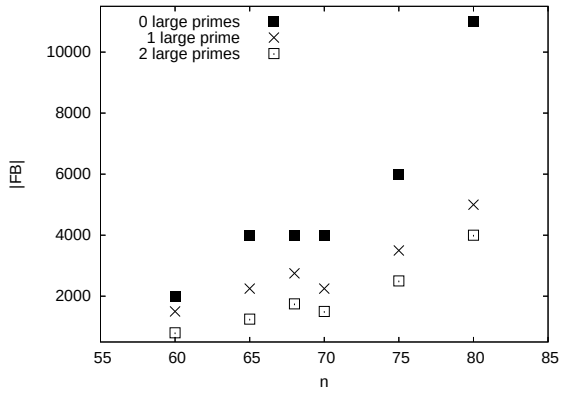


FIGURE 6. Optimal value of $|FB|$ when n varies



4.3. THE RATIO B_2/B_1. Theoretically B_2 should not exceed B_1^2. In practice, when the ratio B_2/B_1 is too high we lose time taking into account partial relations involving primes that are so large that they are very unlikely to occur twice and to lead to a recombination. This phenomenon is known in the context of factorization, and 120 is a common choice of value of B_2/B_1 (see [6]). We ran experiments using 12, 120 and 1200 as values for the ratio B_2/B_1. Figure 7 shows the results for $\Delta = -4(10^{75} + 1)$ with two large primes. We give the optimum timings for each value of the size of the factor base, and compare those values for the three different ratios. It appears that 120 is indeed the best choice, but the performance of the algorithm is not highly dependent on this parameter.

5. COMPUTATIONAL RESULTS

5.1. COMPARATIVE TIMINGS. In Figure 8 we give comparative timings in seconds between no large primes and the large prime variants for discriminants of the form $-4(10^n + 1)$, for n between 60 and 80. We used 2.4GHz Opterons with 16MB of

FIGURE 7. Comparative table of B_2/B_1

$\|FB\|$	12	120	1200
3000	6399.60	6051.11	6173.66
3250	6795.43	6185.67	6754.02
3500	6539.69	6821.77	6754.02
3750	6916.93	6750.88	7456.92
4000	6671.18	6390.48	7009.72

memory, and the NTL library with GMP. It appears that we achieved a significant speed-up by using the large prime strategy. Direct comparison with previous methods based on sieving is hard since the timings available in [10] were obtained on 296 MHz UtraSPARC-II processors; therefore we just quote that the computation of the group structure corresponding to $\Delta = -4(10^{80} + 1)$ took 5.37 days at the time. We also notice that the double large prime variant does not provide an impressive improvement on the overall time for the sizes of discriminant involved. The performance is comparable for discriminants of 60 decimal digits and starts showing an improvement when we reach 75 digit discrimants.

FIGURE 8. Comparative table of the performances

n	0 Large primes	1 Large prime	2 Large primes
60	374	284	280
65	1019	756	776
68	2010	1489	1122
70	2148	1663	1680
75	8409	6669	5347
80	21215	17123	14664

5.2. LARGE DISCRIMINANTS. In the imaginary case, the largest class groups that had been computed using relation collection methods had 90 digits; some 100 decimal digit discriminant class group structures could be computed using the techniques of [21]. With the techniques described in this paper, we achieved the computation of a 110 decimal digit discrimant class group. We used 2.4GHz Opteron processors with 16 GB of memory each for the sieving, and one 2.66 GHz Opteron with 64 GB of memory for the linear algebra, which is the real bottleneck of this algorithm. Indeed, the sieving phase can be trivially parallelized for as many processors as we have and does not require much memory, whereas the linear algebra can only be parallelized into the number of factors of h that we get from Vollmer's algorithm (around 10 in our examples) and requires a lot of memory. Indeed the limit in terms of matrix dimensions for the diophantine solver on a 64GB memory computer seems to be around 10000 columns. For comparison in the case of the 110 decimal digit discriminant we had to handle an 8000-column matrix (after the Gaussian reduction).

ACKNOWLEDGEMENTS

The author thanks Andreas Enge for his support on this project, the fruitful discussions we had and a careful reading of this article. We thank Nicolas Thériault

FIGURE 9. Decomposition of $\mathrm{Cl}(\Delta)$ for $\Delta = -4(10^n + 1)$

n	decomposition
100	$C(2)^5 \times C(4570286810850610233036544737054672922427752 7471)$
110	$C(2)^{11} \times C(8576403641950292891121955131452148838284294200071440)$

and all the organizing comitee of the conference CHILE 2009 where the original results of this paper were first presented. We also thank Jérôme Milan for his support on issues regarding implementation, especially with the TIFA library.

References

[1] E. Bach, *Explicit bounds for primality testing and related problems*, Mathematics of Computations, **55** (1990), 335–380.
[2] J. Buchmann and S. Düllmann, *On the computation of discrete logarithms in class groups*, in "Advances in Cryptology - CRYPTO '90," Lecture Notes in Computer Science, **537** (1991), 134–139.
[3] J. Buchmann and H.C. Williams, *A key-exchange system based on imaginary quadratic fields*, Journal of Cryptology, **1** (1988), 107–118.
[4] S. Cavallar, *Strategies in Filtering in the Number Field Sieve*, in "ANTS-IV: Proceedings of the 4th International Symposium on Algorithmic Number Theory," Lecture Notes in Computer Science, **1838** (2000), 209–232.
[5] H. Cohen, "A course in computational algebraic number theory," vol 138 of Graduate Texts in Mathematics, Springer-Verlag, 1991.
[6] S. Contini "Factoring integers with the self initializing quadratic sieve," Master thesis, University of Georgia, 1997
[7] J.E. Gower and S. Wagstaff, *Square form factorization*, Mathematics of Computations, **77** (2008), 551–588.
[8] J.L. Hafner and K.S. McCurley, *A rigorous subexponential algorithm for computation of class groups*, J. Amer. Math. Soc., **2** (1989), 839–850.
[9] D. Hühnlein, M.J. Jacobson, S. Paulus and T. Takagi, *A cryptosystem based on non-maximal imaginary quadratic orders with fast decryption*, in "Advances in Cryptology - EUROCRYPT '98," Lecture Notes in Computer Science, **1403** (1998), 294–307.
[10] M. Jacobson, "Subexponential Class Group Computation in Quadratic Orders," Ph.D thesis, Technische Universität Darmstadt, 1999, Shaker Verlag GmbH.
[11] M.J. Jacobson and H.C. Williams, "Solving the Pell equation," CMS Books in Mathematics, Springer-Verlag, 2009.
[12] A.K. Lenstra and M.S. Manasse, *Factoring with two large primes (extended abstract)*, in "Advances in Cryptology - EUROCRYPT '90," Lecture Notes in Computer Science, **473** (1991), 72–82.
[13] A.K. Lenstra, *On the calculation of regulators and class numbers of quadratic fields*, in "Journées arithmétiques," Cambridge Univ. Press, (1982).
[14] P.C. Leyland, A.K. Lenstra, B. Dodson, A. Muffett and S. Wagstaff, *MPQS with Three Large Primes*, in "ANTS-V: Proceedings of the 5th International Symposium on Algorithmic Number Theory," Lecture Notes in Computer Science, **2369** (2002), 446–460.
[15] J. Milan, "TIFA", http://www.lix.polytechnique.fr/Labo/Jerome.Milan/tifa/tifa.xhtml.
[16] J. Neukirch, "Algebraic Number Theory," vol 322 of Comprehensive Studies in Mathematics, Springer-Verlag, 1999. Translation into english by Norbert Schappacher.
[17] C. Pomerance, *Analysis and comparison of some integer factoring algorithms*, in "Computational methods in number theory I," Mathematical Centre Tracts, **154** (1982), 89–139.
[18] D. Shanks, *Class number, a theory of factorization, and genera*, in "Proceedings of symposia in pure mathematics," American Mathematical Society, **20** (1969), 415–440.
[19] C.Siegel, *Über die Klassenzahl quadratischer Zahlkörper*, Acta Arithmetica, **1** (1936), 83–86.
[20] A. Storjohann, "IML", http://www.cs.uwaterloo.ca/ z4chen/iml.html.
[21] A. Sutherland, "Order Computations in Generic Groups," Ph.D thesis, Massachusetts Institute of Technology, 2007.

[22] U. Vollmer, *A note on the Hermite Basis Computation of Large Integer Matrices*, in "ISSAC '03: Proceedings of the 2003 international symposium on Symbolic and algebraic computation," ACM, (2003), 255–257.

Received June 2009; revised XXX.

E-mail address: biasse@lix.polytechnique.fr