Distributed class group computation

Johannes Buchmann
Stephan Düllmann
FB 14-Informatik
Universität des Saarlandes
D-6600 Saarbrücken
Germany

Abstract

We present an improved sequential and a parallel version of the algorithm of Hafner and McCurley for the computation of the class group of imaginary quadratic fields. We describe the implementation of this algorithm on a network of UNIX-workstations using the system LIPS.

1 Introduction

In [7] Hafner and McCurley present a probabilistic algorithm which under the assumption of the generalized Riemann hypothesis (GRH) computes the class number and the structure of the class group of an imaginary quadratic field in expected running time $L(D)^{\sqrt{2}+o(1)}$ where D is the absolute discriminant of the field and where $L(D) = \exp(\sqrt{\log D \log\log D})$.

In [3] we describe a practical version of that algorithm and we give several numerical examples and timings for discriminants up to 40 decimal digits.

In this paper we present an improved sequential version and a parallel version of the class group algorithm and we describe its implementation using the system LIPS which is described in [2]. Numerical examples demonstrate how the new algorithm drastically reduces the effective running time of the algorithm.

In section 2 we briefly describe the idea of the algorithm. In section 3 we describe in detail the most time consuming part of the algorithm and we present some refinements for the sequential algorithm. In section 4 we show how to parallelize the algorithm. In section 5 we give computational results for a 55-digit discriminant.

2 The algorithm

2.1 The general idea

We briefly describe the main idea of the algorithm of Hafner and McCurley [7]:

Let $\mathcal{K}$ be an imaginary quadratic field of absolute discrimant D, Cl its class group and h its class number. Let

$$F = \{\wp_1, \ldots, \wp_k\}$$

be a set of prime ideals of $\mathcal{K}$ generating Cl. This set can easily be computed, since under GRH the class group Cl of $\mathcal{K}$ is generated by the prime ideals of norm below $6(\log D)^2$ (see [10] and [1]).

We define the mapping

$$\begin{array}{rccl} \varphi : & \mathbb{Z}^k & \rightarrow & Cl \\ & (e_1, \ldots e_k) & \mapsto & \prod_{i=1}^{k} \wp_i^{e_i} \ . \end{array}$$

Obviously φ is a homomorphism and, since F generates Cl, φ is surjective. The kernel of φ

$$L = \ker\varphi = \left\{ (e_1, \ldots e_k) \ : \ \prod_{i=1}^{k} \wp_i^{e_i} = 1 \right\}$$

is a k-dimensional lattice in $\mathbb{Z}^k$ and we have

$$\mathbb{Z}^k/L \cong Cl$$

and so

$$\det L = \#Cl = h.$$

The basic idea of the algorithm is to find vectors in L (so-called *relations*) by means of a probabilistic method until the full lattice is generated.

Assuming GRH and using the analytic class number formula an approximation h^* to the class number h with

$$h^* < h < 2h^* \tag{1}$$

can be computed in polynomial time (see [5]). Therefore, we have an easy criterion to decide whether the current set of relations generates the full lattice L: Each set of relations generates a sublattice L' of L. If L' has dimension k we know $h \mid h' = \det L'$. So if $h' < 2h^*$ we know $L' = L$ and $h' = h$.

After having computed h, the structure of the class group, i.e. integers m_i $(1 \le i \le k)$ such that

$$\begin{aligned} &m_i \mid m_{i+1} \qquad (1 \le i \le k-1), \\ &h = \prod_{i=1}^{k} m_i, \\ &Cl \cong \bigoplus_{i=1}^{k} \mathbb{Z}/m_i\mathbb{Z}, \end{aligned}$$

can be found easily by computing the Smith normal form (SNF) of a basis of L. For details see [7].

Moreover, if we keep track of the unimodular transformations occuring in the computation of the SNF, we obtain a minimal generating set for the class group in terms of the elements of F.

2.2 The implementation

The general idea described above and our practical experience lead to the following algorithm:

Algorithm 2.1

Input: *Absolute discriminant D*
Output: *Class number h and structure and minimal generating set of the class group Cl.*

(1) Compute the set $F = \{\wp \ : \ N(\wp) \le 6(\log D)^2\} = \{\wp_1, \dots, \wp_k\}$ of prime ideals generating Cl and the approximation h^* satisfying (1).

(2) Find at least k vectors of L generating a sublattice L' of L.

(3) Compute the dimension k' of L'. If $k' = k$ compute the determinant h' of L'.

(4) If $k' = k$ and $h' < 2h^*$ go to step (5) else find an additional vector of L to enlarge the sublattice L' and repeat step (3).

(5) Compute the structure and a minimal generating set of Cl.

Here are some remarks concerning the details of the algorithm:

Step (1): The details are given in [3].

Step (2): The computation of the relations is the most time consuming part of the algorithm. The version presented in [3] has the following features:

- We generate a sparse relation matrix, since otherwise the computation of the determinant is infeasible.
- We generate $(k+5)$ relations. Then the determinant, that is computed in step (3), is only a small multiple of h. In many cases, we already find the class number in this step; step (4) does not have to be executed.

Step (3): The computation of the determinant of L' is done by computing the Hermite normal form (HNF) of the relation matrix. There are two stages:

(3.1) Perform a slightly modified version of the structured Gaussian elimination algorithm [8] to decrease the size of the matrix. Here we use the sparsity of the relation matrix.

(3.2) By Gaussian elimination compute a multiple Δ of the lattice determinant and then perform modular Hermite reduction [6] modulo Δ. For details see [3].

In very few cases it turns out in step (3) that the lattice found in step (2) is not of dimension k.

Step (4): The additional relations are found by the same technique as in step (2). The computation of the new determinant is done by updating the HNF using standard techniques as described in [6].

Step (5): For the computation of the SNF of the basis of L we use standard techniques (see [7]). Since the size of the matrix to be considered is very small, no special treatment is necessary.

3 Improvements of the sequential algorithm

3.1 Finding one relation

First of all we describe the general idea how to find one random relation in L. This method is a modification of an algorithm of Seysen presented in [11].

Let $B \in \mathbb{Z}_{>0}$ be fixed. According to a heuristic strategy described below we pick a vector $\underline{v} = (v_1, \ldots v_k) \in \{0, \ldots B\}^k$ and compute the reduced ideal $\mathbf{a}'$ in the class of

$$\mathbf{a} = \prod_{i=1}^{k} \wp_i^{v_i}.$$

Then we try to factor $\mathbf{a}'$ over the set F. If there is a factorization

$$\mathbf{a}' = \prod_{i=1}^{k} \wp_i^{v_i'}$$

with $v_i' \in \mathbb{Z}$ $(1 \leq i \leq k)$ then the ideal

$$\mathbf{a}(\mathbf{a}')^{-1} = \prod_{i=1}^{k} \wp_i^{v_i - v_i'}$$

is principal and so the vector $\underline{e} = \underline{v} - \underline{v}'$ is a relation.

A practical algorithm for the computation of power products of ideals has been presented in [4], see also [12].

A way of factoring the ideal $\mathbf{a}'$ over F can be derived from [11], Theorem 3.1. The main work is to find a factorization of $N(\mathbf{a}')$ over the set $\hat{F}$ of primes built by the norms of the ideals in F.

3.2 Finding the relation matrix

We present a heuristic strategy for finding the relation matrix. This strategy is a modification of the method described in [3] and is very efficient in practice. We define the set

$$K = \{i \ : \ 1 \leq i \leq k, \ N(\wp_i) < \log D, \ \wp_i \text{ not ambigious}\}$$

and we let $k_0 = \#K$, $k_1 = k_0/2$, $B = 30$. The size of the parameters k_1 and B varies with the size of the discriminant. The values chosen here have experimentally turned out to be optimal for a 55-digit discriminant. The vectors $\underline{v}$ used to generate the relations are chosen from the set

$$V = \left\{ \underline{v} = (v_1, \ldots v_k) \in \mathbb{Z}^k \ : \ \sum_{i=1}^{k_0} |\operatorname{sign} v_i| = k_1, \ \sum_{i=k_0+1}^{k} | v_i | \leq 1, \ 0 \leq v_i \leq B, \ 1 \leq i \leq k \right\}.$$

Those vectors have k_1 non zero entries in the first k_0 positions and at most one non zero entry on the last $k - k_0$ positions. Using such vectors has two advantages:

- The corresponding power products can be computed very quickly if the powers ${\wp_i}^j,\ 1 \le i \le k_0,\ 1 \le j \le B$ have been precomputed.
- The resulting relations matrix is very sparse.

The non zero entry on the last $k - k_0$ positions is used to make the relation matrix very likely to be non singular.

Here is the algorithm for finding the relation matrix:

Algorithm 3.1

(1) Determine K and k_0.

(2) Precompute the ideal powers $\mathbf{a}_{ij} = {\wp_j}^i$ for $j \in K$, $1 \le i \le B$.

(3) Set $n = 0$ (number of relations already found) and set $m_i = 0$ $(1 \le i \le k)$ (vector to mark the positions already represented in the relation matrix).

(4) If $n \ge k + 5$ (enough relations are found) and $m_i = 1$ for $1 \le i \le k$ (all positions are marked) then the algorithm terminates.

(5) Choose at random pairwise different positions $\nu_1, \dots \nu_{k_1} \in K$ and exponents $v_{\nu_1}, \dots v_{\nu_{k_1}} \in \{1, \dots B\}$.
Set $v_i = 0$ for $1 \le i \le k,\ i \notin \{\nu_1, \dots \nu_{k_1}\}$.
If the set $J = \{i\ :\ 1 \le i \le k,\ m_i = 0,\ ,\ i \ne \nu_j \text{ for } 1 \le j \le k_1\}$ is not empty, choose at random a position $\mu \in J$ and set $v_\mu = 1$.

(6) Compute the reduced ideal $\mathbf{a}'$ in the class of $\mathbf{a} = \prod_{i=1}^{k} \wp_i^{v_i}$.

(7) Try to find a factorization $\mathbf{a}' = \prod_{i=1}^{k} \wp_i^{v_i'}$.
If there is no such factorization go back to step (5).

(8) Compute the relation vector $\underline{e} = \underline{v} - \underline{v}'$.

(9) Find a position $i \in \{1, \dots k\}$ with $m_i = 0$ and $e_i \ne 0$. If there is one, set $m_i = 1$.
Set $n = n + 1$ and go to step (4).

The computation of the power product in step (6) is done by means of Shanks algorithms NUCOMP and NUDUPL [13] which yield a considerable speedup compared to the standard methods. The factorization process (step (7)) is speeded up by a modification of the early abort strategy (see e.g. [9]).

3.3 Large prime relations

Another speed up of the algorithm is achieved by using large prime relations. The idea of collecting large prime relations is well known from integer factoring algorithms (see e.g. [9]). It can be adopted for our purposes.

In step (7) of Algorithm 3.1 we have to test whether there is a complete factorization of $\mathbf{a}'$ over F. Now assume we find two incomplete factorizations with the same cofactor $\wp'$:

$$\mathbf{a}'_j = \prod_{i=1}^{k} \wp_i^{v'_{ij}} \cdot \wp'^{w'_j}$$

for $j = 1, 2$ where $\wp' \notin F$ is a prime ideal and $w'_1, w'_2 \in \{\pm 1\}$. Then we can construct a new relation vector from the equation

$$1 = (\mathbf{a}_1 \cdot (\mathbf{a}'_1)^{-1})(\mathbf{a}_2 \cdot (\mathbf{a}'_2)^{-1})^{w'_1 w'_2} = \prod_{i=1}^{k} \wp_i^{(v_{i1}-v'_{i1})+w'_1 w'_2 (v_{i2}-v'_{i2})}.$$

This idea can be used to modify Algorithm 3.1 as follows: Whenever we find an incomplete factorization with a prime cofactor $\wp'$ satisfying $N(\wp') \leq C$ for an appropriately chosen constant $C < (\log D)^4$ we store that factorization. If another incomplete factorization with the same cofactor is found, we have a new relation. We currently use $C = 10(\log D)^2$ and are experimenting to find out optimal values for C for different sizes of D.

This means that step (7) of the algorithm is replaced by two steps:

(7.1) Try to find a factorization of $\mathbf{a}'$ over F. If there is a complete factorization goto step (8). If there is a large prime factorization with $N(\wp') \leq C$ go to step (7.2). Else go back to step (5).

(7.2) If the the cofactor $\wp'$ also occures as a cofactor in a large prime factorization that was found previously compute the new relation and go to step (8) else store $\wp'$, $\mathbf{a}'$, $\underline{v}$ and $\underline{v}'$ in the list of large prime factorizations.

4 Parallelization

4.1 The general idea

In this section we describe a parallel version of the class group algorithm that can be implemented using the LIPS-system for distributed computation [2]. The LIPS-system works as follows:

There is one *host computer* which manages the distribution and puts the results together and an indefinite number of *client computers* which perform the actual computation.

The LIPS-system has two important features which influence the design of the parallel algorithm:

- The system only uses the idle time of the client computers. So one cannot be sure that a certain amount of CPU time is available on a certain machine.
- The LIPS system uses the standard computer network Ethernet. So there is no memory sharing and there should not be much communication between the clients.

Algorithm 3.1 for constructing the relation matrix consists of a big loop (steps (5) to (9)). In each iteration the algorithm checks whether a random vector yields a relation. These iterations are relatively independant of each other and therefore this part of the algorithm is well suited for distributed computation. So in our algorithm the clients find the relations and large prime factorizations.

Since large prime factorizations having the same cofactor may be found on different clients there is a need of communication to combine large prime factorizations to relations. So relations and large prime factorizations have to be sent from the clients to the host and handled there.

Moreover, the vector $(m_1, \ldots m_k)$, which carries information about the factor basis elements that are already represented in the relation matrix, has to be updated on the host. Since changes to this vector influence the choice of the random vectors $\underline{v}$, those changes have to be sent from the host to the clients.

So the communication complexity of this algorithm is higher than for the Multiple polynomial quadratic sieve factoring algorithm.

4.2 The algorithms

The ideas mentioned above lead to the following algorithms:

Algorithm 4.1 (Host)

(1) Start the clients, send D and the set F to them.

(2) Set $n = 0$ and $m_i = 0$ $(1 \leq i \leq k)$.

(3) If $n \geq k + 5$ (enough relations are found) and $m_i = 1$ for $1 \leq i \leq k$ (all positions are marked) stop all clients and terminate the algorithm.

(4) Wait for relations or large prime factorizations sent from the clients. If a relation is received go to step (6). If a large prime factorization is received go to step (5).

(5) If the prime ideal $\wp'$ occures as a cofactor in the stored list of large prime factorizations compute the new relation and go to step (6) else store the large prime factorization in the list and go back to step (4).

(6) Find a position $i \in \{1, \ldots k\}$ with $m_i = 0$ and $e_i \neq 0$. If there is one set $m_i = 1$ and send i to all clients.

Algorithm 4.2 (Client)

(1) Determine K and k_0.

(2) Precompute the ideal powers $\mathbf{a}_{ij} = {\wp_j}^i$ for $j \in K,\ 1 \leq i \leq B$.

(3) Set $m_i = 0$ for $1 \leq i \leq k$.

(4) If there is a message i from the host, set $m_i = 1$.

(5) Choose the random exponent vector $\underline{v}$ as described in step (5) of Algorithm 3.1.

(6) Compute the reduced ideal $\mathbf{a}'$ in the class of $\mathbf{a} = \prod_{i=1}^{k} \wp_i^{v_i}$.

(7) Try to find a factorization of $\mathbf{a}'$ over F. If there is a complete factorization or a large prime factorization with $N(\wp) \leq C$ send the corresponding data to the host.

(8) Go back to step (4).

5 Computational results

As an example for the efficiency of our parallel algorithm we give computational results for a 55-digit discriminant. We computed

Table 5.1

absolut discriminant d	$4 \cdot 10^{54} + 4$
size k of the factor basis	4585
class number h	1056175002108254379317829632
structure of the class group $(m_i,\ m_i \neq 1,\ 1 \leq i \leq k)$	2, 2, 2, 2, 2, 33005468815882949353682176

In Table 5.2 below we list the data for the parallel computation of the relation matrix on 14 clients (Algorithm 4.2). The computation started on November 30th at 13.49 and was finished on December 12th at 7.58. So the real running time was 282.2 hours.

In the first column we give the name of the clients. In the second column we give the CPU-time used on the clients. In the third column we give the number of vectors $\underline{v}$ which were tested. In the fourth column we give the number of successes, i.e. relations or large prime factorizations sent from the clients to the host.

Table 5.2

client name	CPU time (in hours)	number of trials ($\cdot 10^6$)	number of successes
crypt1	149.4	3.18	665
crypt2	213.8	4.58	953
cauchy	281.9	6.91	1412
fermat	279.8	6.86	1390
fourier	85.3	2.17	423
gauss	281.1	6.89	1395
hilbert	281.2	6.86	1392
laplace	276.2	6.78	1363
riemann	281.5	6.86	1431
batman	234.3	4.83	983
robin	125.3	2.46	529
p1sun	212.8	5.37	1037
p2sun	125.2	3.07	635
cadsun	104.5	2.69	523

Algorithm 4.1 (host) was also performed on one of the workstations mentioned above (*crypt1*). Its CPU time was 5.7 hours.

Concluding this description we list the running times of all steps of Algorithm 2.1:

Table 5.3

	CPU time
step (1)	7 seconds
step (2)	see above
step (3)	27.3 hours
step (4)	not performed
step (5)	3 seconds

References

[1] E. Bach, *What to do until the witness comes*, preprint 1988.

[2] J. Buchmann, M. Diehl, R. Roth, *LIPS - a system for distributed applications*, submitted to ISSAC '91.

[3] J. Buchmann, S. Düllmann, *A probabilistic class group and regulator algorithm and its implementation*, to appear in: Proceedings of the Colloquium on Computational Number Theory (Debrecen 1989).

[4] J. Buchmann, S. Düllmann, H.C. Williams, *On the complexity and efficiency of a new key exchange system*, Proceedings EUROCRYPT'89, to appear.

[5] J. Buchmann, H.C. Williams, *On the computation of the class number of an algebraic number field*, Math. Comp. **53** (1989), 679-688.

[6] P.D. Domich, R. Kannan, L.E. Trotter, *Hermite normal form computation using modulo determinant arithmetic*, Math. Op. Res. **12** (1987), 50-59.

[7] J.L. Hafner, K.S. McCurley, *A rigorous subexponential algorithm for computation of class groups*, Journal AMS, to appear.

[8] B.A. LaMacchia, A.M. Odlyzko, *Computation of discrete Logarithms in Prime Fields*, to appear.

[9] C. Pomerance, *Analysis and Comparison of some Integer Factoring Algorithms*, Computational Methods in Number Theory (R. Tijdeman and H.W. Lenstra, Jr., eds.), Mathematisch Centrum Tract 154, Amsterdam, 1982, 89-139.

[10] R. Schoof, *Quadratic fields and factorization*, Computational Methods in Number Theory (R. Tijdeman and H.W. Lenstra, Jr., eds.), Mathematisch Centrum Tract 154, Amsterdam, 1982, 235-286.

[11] M. Seysen, *A Probabilistic Factorization algorithm with quadratic Forms of Negative Discriminant*, Math. Comp. **48** (1987), 737-780.

[12] D. Shanks, *Class number, a theory of factorization and genera*, Proc. Symp. Pure Math. **20** (1969 Institute on number theory), Amer. Math. Soc., Providence, 1971, 415-440.

[13] D. Shanks, *On Gauss and Composition*, Number Theory and Applications (R.A. Mollin, ed.), Kluwer Academic Publishers, 1989, 163-204.