MATHEMATICS OF COMPUTATION
Volume 76, Number 260, October 2007, Pages 2139–2160
S 0025-5718(07)01935-7
Article electronically published on April 19, 2007

A FAST, RIGOROUS TECHNIQUE FOR COMPUTING THE REGULATOR OF A REAL QUADRATIC FIELD

R. DE HAAN, M. J. JACOBSON, JR., AND H. C. WILLIAMS

Abstract. We present a new algorithm for computing the regulator of a real quadratic field $\mathbb{Q}(\sqrt{D})$, based on an algorithm for unconditionally verifying the correctness of the regulator produced by a subexponential algorithm, that runs in expected time $O(D^{1/6+\epsilon})$ under the Generalized Riemann Hypothesis. The correctness of our algorithm relies on no unproven hypotheses and is currently the fastest known unconditional algorithm for computing the regulator. A number of implementation issues and performance enhancements are discussed, and we present the results of computations demonstrating the efficiency of the new algorithm.

1. Introduction

Many problems in mathematics are related to determining the *fundamental unit* η_0 (> 1) of a related real quadratic number field $K = \mathbb{Q}(\sqrt{D})$. Perhaps one of the most well-known examples is the problem of solving the Pell equation $x^2 - Dy^2 = 1$ in integers x and y for a certain nonsquare integer D, which is described in detail in Williams [25], but other examples can be found in Lenstra [15] and Jacobson et al. [9]. In most cases, the fundamental unit grows at an exponential rate as the discriminant Δ of K increases, and as a result, it is difficult to work with it explicitly. Therefore it is often easier to work with the *regulator* R of K, which is defined to be the natural logarithm of the fundamental unit, or with the (more or less equivalent) base 2-logarithm of the fundamental unit, which we denote by R_2.

A number of techniques have been developed that can be used to compute the regulator of a real quadratic number field. In particular, there was an important breakthrough in 1987, when A.K. and H.W. Lenstra introduced the basic idea behind a method for factoring integers of subexponential run time complexity in [12] and later in [13] based on properties of the class group of an imaginary quadratic order. It should be mentioned that some of the ideas behind this technique, which is essentially an index calculus method, were mentioned earlier by Pohst and Zassenhaus [18]. This method depends on the assumption of a Generalized Riemann Hypothesis (GRH) for its run time complexity. McCurley [17] and Hafner and McCurley [7] first used the underlying idea of this algorithm to construct a

Received by the editor May 23, 2005.
2000 *Mathematics Subject Classification.* Primary 11Y40; Secondary 11Y16.
Key words and phrases. Regulator computation, regulator verification, real quadratic number fields, reduced principal ideals, (f, p) representations.
The second author's research is supported by NSERC of Canada.
The third author's research is supported by NSERC of Canada and iCORE of Alberta.

subexponential technique for determining the structure of the class group of an imaginary quadratic field. Buchmann [2] was the first to point out that the subexponential technique can also be applied to the problem of computing a system of fundamental units and class group of an arbitrary number field, after which his ideas were modified and implemented by Cohen, Diaz y Diaz and Olivier [3, 4]. The complexity analysis of computing the regulator for a real quadratic field using this technique was improved by Abel [1] in her doctoral thesis, after which the complexity analysis was again improved by Vollmer [23]. Like previous versions, the complexity of Vollmer's algorithm is dependent on the truth of the GRH; under the GRH, the expected time to compute the regulator is $O(e^{(\sqrt{2}+o(1))(\log D \log\log D)^{1/2}})$.

Unfortunately, the correctness of the output of these methods also depends on the unproven GRH, and many mathematical problems that can easily be solved as soon as a good approximation of the regulator of a related real quadratic field is computed (see [9] for one such problem) require this computed value to be unconditionally correct. For example, given the regulator, one can compute a representation of the fundamental unit and use Lucas sequences to compute the coefficients of any desired unit in the field. If the regulator is not unconditionally correct, there is no guarantee that the corresponding unit is fundamental, and one cannot obtain all units in the field.

Up to the publication of [9], the fastest known unconditional method for computing the regulator of a real quadratic number field was the algorithm with run time complexity $O(D^{1/5+\epsilon})$ under the GRH that is described in Lenstra [14]. The first step of the algorithm uses analytic techniques to obtain an approximation of hR, where h is the class number of the field. Then, the ideal with distance closest to this approximation is computed, and baby steps and giant steps are used to determine an approximation R' of a multiple of R. After this, baby steps and giant steps are used to determine a lower bound for R. Finally, it is checked for all primes q that can possibly divide the multiplier whether the ideal with distance closest to R'/q is equal to the ring of integers $\mathcal{O}_K$, in which case q divides the multiplier. Using these primes the full multiplier is determined, after which it is easy to determine an approximation for R.

As observed in [9], the key to obtaining a run time complexity of $O(D^{1/6+\epsilon})$ is that we can use subexponential techniques rather than baby steps/giant steps to compute the multiple of R. The output of the subexponential algorithms is unconditionally a multiple of R, and under the GRH is equal to R. Given this multiple, one can use the second and third stages of the $O(D^{1/5+\epsilon})$ algorithm to determine the multiplier and thus compute R, resulting in an unconditionally correct algorithm for computing R with expected run time complexity $O(D^{1/6+\epsilon})$ under the GRH.

In this paper, we describe in detail the $O(D^{1/6+\epsilon})$ algorithm for unconditionally computing the regulator of a real quadratic field. We first state basic results on ideals and infrastructure in real quadratic fields that are required for our algorithm in Section 2. We use (f,p)-representations as described in [11] for arithmetic in the infrastructure. This technique, which has the advantages of faster algorithms and approximations of distances with relatively low precision, is briefly summarized in Section 3, together with the relevant algorithms. The regulator verification algorithm is described and analyzed in terms of run time complexity in Section 4 and a number of practical improvements described in detail in de Haan [5], including

a parallelization of the baby step/giant step algorithm, are outlined in Section 5. In order to make our algorithm truly rigorous, it is essential that we do a full precision analysis on it, as the regulator is a transcendental number and can only be computed up to some finite precision. This analysis has been performed in detail in [5], but the main results are stated in Section 6. Finally, numerical results comparing the performance of our algorithm to the $O(D^{1/5+\epsilon})$ algorithm and demonstrating its efficiency are presented in Section 7. In particular, we were able to unconditionally compute the regulator of a real quadratic field with a 65 decimal digit discriminant, a significant improvement to what was previously possible.

2. Basics

We precede our discussion by listing some well-known facts about real quadratic number fields. The information from this section can be found in greater detail in Williams and Wunderlich [26] and in van der Poorten et al. [22].

2.1. **Continued fractions.** Every $\phi \in \mathbb{R}_{>0}$ can be written as a regular continued fraction

$$q_0 + \cfrac{1}{q_1 + \cfrac{1}{\ddots + \cfrac{1}{q_{n-1} + \cfrac{1}{\phi_n}}}}$$

with $q_i \in \mathbb{Z}$ and $q_i \geq 1$ when $i \geq 1$, which we denote by

$$\phi = \langle q_0, q_1, q_2, \ldots, q_{n-1}, \phi_n \rangle = \langle q_0, q_1, q_2, \ldots, q_{n-1}, \ldots \rangle$$

for any $n \in \mathbb{Z}_{>0}$. We can compute the values for ϕ_i and a_i recursively by putting $\phi_0 := \phi$, $a_i = \lfloor \phi_i \rfloor$ and $\phi_{i+1} := \frac{1}{\phi_i - q_i} > 1$ when $i \geq 0$. The variables q_i $(i \geq 0)$ are called the *partial quotients* and the $C_n = \langle q_0, q_1, q_2, \ldots, q_n \rangle$ the *convergents* of the continued fraction of ϕ. The convergents are given by $C_n = A_n/B_n$, where A_i and B_i are defined by $A_{-2} = 0$, $B_{-2} = 1$, $A_{-1} = 1$, $B_{-1} = 0$ and

$$\begin{aligned} A_i &= q_i A_{i-1} + A_{i-2}, \\ B_i &= q_i B_{i-1} + B_{i-2} \end{aligned}$$

for $i = 0, 1, \ldots$.

Now, let D be a positive squarefree number and assume that we have $P, Q \in \mathbb{Z}$ such that Q divides $P^2 - D$. Then we can determine the partial quotients q_i of the irrational number $\phi = \phi_0 = (P + \sqrt{D})/Q = \langle q_0, q_1, \ldots, q_{n-1}, \phi_n \rangle$ by making use of the formulas

$$\begin{aligned} P_{i+1} &= q_i Q_i - P_i, \\ Q_{i+1} &= (D - {P_{i+1}}^2)/Q_i = Q_{i-1} - q_i(P_{i+1} - P_i), \\ q_{i+1} &= \lfloor (P_{i+1} + \sqrt{D})/Q_{i+1} \rfloor = \lfloor (P_{i+1} + d)/Q_{i+1} \rfloor, \end{aligned}$$

where $d = \lfloor \sqrt{D} \rfloor$, $P_0 = P$, $Q_0 = Q$ and $q_0 = \lfloor \phi_0 \rfloor$. Also,

$$\phi_n = (P_n + \sqrt{D})/Q_n \ .$$

If we put $\Psi_1 = 1$ and define

$$\Psi_k = \prod_{i=1}^{k-1} \psi_i \ (k > 1)$$

where

$$\psi_i = (-\overline{\phi}_i)^{-1} = (P_i + \sqrt{D})/Q_{i-1},$$

then

$$\Psi_i = A_{i-2} - \overline{\phi}_0 B_{i-2} \ .$$

In addition, for $i \geq 1$ we have that

$$N(\Psi_i) = \Psi_i \overline{\Psi}_i = (-1)^{i-1} \frac{Q_{i-1}}{Q_0}$$

and that

$$\Psi_{i+2} = q_i \Psi_{i+1} + \Psi_i \ . \tag{2.1}$$

2.1.1. *Reduced ideals and reduction.* Let $[\alpha, \beta]$ denote the $\mathbb{Z}$-module $\alpha\mathbb{Z} + \beta\mathbb{Z} = \{x\alpha + y\beta \mid x, y \in \mathbb{Z}\}$. If we define ω to be $(1 + \sqrt{D})/2$ when $D \equiv 1 \pmod 4$ and to be $\sqrt{D}$ otherwise, then we have $\mathcal{O}_K = [1, \omega]$, where $\mathcal{O}_K$ is the ring of integers in K. In fact, we have that every ideal in $\mathcal{O}_K$ can be written as $\mathfrak{a} = [a, \beta]$, where β is of the form $b + c\omega$, $a, b, c \in \mathbb{Z}$, $c \mid a$, $c \mid b$, (the *norm* of $\mathfrak{a}$) $N(\mathfrak{a}) = ac$ and $N(\mathfrak{a}) \mid \beta\overline{\beta}$. If $c = 1$, we call the ideal $\mathfrak{a}$ *primitive*. A primitive ideal $\mathfrak{a}$ is said to be *reduced* if $N(\mathfrak{a})$ is a minimum in $\mathfrak{a}$, which is exactly when there is no nonzero element $\alpha \in \mathfrak{a}$ such that both $|\alpha| < N(\mathfrak{a})$ and $|\overline{\alpha}| < N(\mathfrak{a})$. We use the usual notation (α) to denote the principal ideal generated by $\alpha \in \mathcal{O}_K$.

From these observations we can deduce (see [26]) that every primitive ideal $\mathfrak{a}$ of $\mathcal{O}_K$ can be written as $[Q/\sigma, (P + \sqrt{D})/\sigma]$ for some $P, Q \in \mathbb{Z}$ where $\sigma = 2$ if $D \equiv 1 \pmod 4$ and $\sigma = 1$ otherwise. Furthermore, we must have $Q|(D - P^2)$. We use the notation $\mathfrak{a} = (Q, P)$ from here on.

Now, if we put $\phi = \phi_0 = (P + \sqrt{D})/Q$, it can be shown that we can produce a sequence of primitive ideals (Q_i, P_i) equivalent to $\mathfrak{a}$ by applying the continued fraction algorithm to ϕ. If we start this process with a (not necessarily reduced) primitive ideal $\mathfrak{a}_1 = (Q_0, P_0)$ we eventually obtain a reduced ideal $\mathfrak{a}_j = (Q_{j-1}, P_{j-1}) = (\Psi_j)\mathfrak{a}_1$ in the sequence, after which all of the following ideals (Q_i, P_i) are also reduced. It is easy to show that, for a reduced ideal $\mathfrak{a}_i$, we have that $\psi_i > 1$ and $\psi_i N(\mathfrak{a}_i) = |(P_i + \sqrt{D})/Q_0| < \sqrt{\Delta}$, so that in particular $\psi_i < \sqrt{\Delta}$ and $N(\mathfrak{a}_i) < \sqrt{\Delta}$.

If the initial ideal $\mathfrak{a}_1$ is not reduced we find the first reduced ideal (Q_j, P_j) after $O(\log(Q_0/\sqrt{D}))$ steps, precisely when $Q_j > 0$ and $0 < P_j < \sqrt{D}$ for the first time. After this happens, these conditions are met for the subsequent P_i and Q_i as well, i.e., the subsequent P_i and Q_i become bounded. Therefore, since $\sqrt{D}$ is an irrational number, the sequence $\{q_i\}$ must become periodic. In addition, it can be shown that the corresponding cycle of ideals contains exactly all of the reduced ideals in $\mathcal{O}_K$ equivalent to $\mathfrak{a}_1$, which means that the preperiod represents the steps required to reduce the ideal.

If we start with a reduced ideal $\mathfrak{a}_1$ (so that the cycle starts with the first ideal) that is ambiguous, i.e., such that $\mathfrak{a}_1 = \overline{\mathfrak{a}}_1$, we have a special symmetry property for the ideals (see [22]) in the cycle that says that $\overline{\mathfrak{a}}_i = \mathfrak{a}_{\pi+2-i}$ for all i with $1 \leq i \leq \pi$, where π is the length of the period. This turns out to be very useful later on.

Finally, we state the following theorem, which can be used when we have two reduced ideals $\mathfrak{a}_i = (\Psi_i)\mathfrak{a}_1$ and $\mathfrak{a}_{i+m} = (\Psi_{i+m})\mathfrak{a}_1$ in the cycle of reduced principal ideals. It is either used to determine an upper bound for Ψ_i or to determine a lower bound for Ψ_{i+m} and will be employed frequently in the sequel, for example to get an indication of how many steps need to be taken in the cycle before the generator grows by a certain factor. The proof of this theorem and some other theorems in following sections are omitted, but can be found in [5].

Theorem 2.1. $\Psi_{i+m} > F_{m+1}\Psi_i$, *where* $m \geq 1$, $F_0 := 0$, $F_1 := 1$ *and* $F_n := F_{n-1} + F_{n-2}$ *for* $n \geq 2$.

2.2. **Infrastructure.** When the ideal $\mathfrak{a}_1$ that we use as initial input for the continued fraction algorithm is equal to $(1) = \mathcal{O}_K = [1, \omega]$, which is principal and reduced, we get $\mathfrak{a}_i = (\Psi_i)\mathfrak{a}_1 = (\Psi_i)$, and the cycle of ideals contains all of the reduced principal ideals in $\mathcal{O}_K$. Since in this case $\psi_i > 1$ for all $i \geq 1$, we can see that $\{\Psi_i\}$ is a strictly increasing sequence. After computing all of the ideals in the cycle we find $\mathfrak{a}_{\pi+1} = \mathfrak{a}_1$ and $(\Psi_{\pi+1}) = (\Psi_1) = (1)$, where π is again the length of the cycle, which indicates that $\Psi_{\pi+1}$ must be a unit. In fact, $\Psi_{\pi+1}$ is the fundamental unit of $\mathcal{O}_K$.

We define the *distance* δ_i *from* $\mathfrak{a}_1$ *to* $\mathfrak{a}_i$ by $\delta_i = \log \Psi_i$ when $\mathfrak{a}_i = (\Psi_i)\mathfrak{a}_1$. When we discuss the *distance of* an ideal $\mathfrak{a}_j$, this refers to the distance from (1) to $\mathfrak{a}_j$, where we start our continued fraction algorithm with the ideal $\mathfrak{a}_1 = [1, \omega]$. The distance is used as an indication of how far around the cycle of reduced principal ideals an ideal is located.

By Levy's Law (see [16]) we expect that $\lim_{i\to\infty} \sqrt[i]{\Psi_i} = e^{\tau}$ with $\tau \approx 1.186$. If i is relatively large and we take the natural logarithm on both sides, we find that

$$\ln \sqrt[i]{\Psi_i} \approx \ln e^{\tau} \Rightarrow \frac{1}{i} \ln \Psi_i \approx \tau$$
$$\Rightarrow i \approx \frac{\ln \Psi_i}{\tau} \approx 0.843 \ln \Psi_i \ .$$

Since $\ln \Psi_{\pi+1} = \ln \eta_0 = R$, this shows us that we would expect that the regulator would provide us with a fairly good estimate of the number of (reduced) ideals in the cycle. Even though Levy's Law is probabilistic, computations have tended to confirm its accuracy, particularly when the regulator is large (see, for example, [24]). Similarly, taking the base 2-logarithm on both sides, we have that

$$\log \Psi_i \approx i\tau \log e \approx 1.7i, \tag{2.2}$$

which shows us that there is an approximately linear relation between the subscript of an ideal and its distance.

Assume that we have two ideals $\mathfrak{a}_i$ and $\mathfrak{a}_j$ as a result of applying the continued fraction algorithm on $\mathfrak{a}_1 = [1, \omega]$. We want to investigate the product $\mathfrak{a}_i\mathfrak{a}_j$, which by definition is the ideal generated by the products of elements in $\mathfrak{a}_i$ and $\mathfrak{a}_j$. Both ideals are principal, so if $\mathfrak{a}_i = \Psi_i\mathcal{O}_K$ and $\mathfrak{a}_j = \Psi_j\mathcal{O}_K$ it is easy to verify that their product must be $(\Psi_i\Psi_j)\mathcal{O}_K = (\Psi_i\Psi_j)$, which is principal as well. Even though both $\mathfrak{a}_i$ and $\mathfrak{a}_j$ are reduced, their product need not be. However, it is possible to write $\mathfrak{a}_i\mathfrak{a}_j = (u)\mathfrak{b}_1$, where $u \in \mathbb{Z}$ and $\mathfrak{b}_1$ is primitive, and if we apply the continued fraction algorithm to $\mathfrak{b}_1$ in order to produce a sequence $\mathfrak{b}_j$ of ideals with $\mathfrak{b}_j = (\Psi'_j)\mathfrak{b}_1$, we eventually find a least index k where $\mathfrak{b}_k$ is reduced. Then $\mathfrak{b}_k = (\Psi'_k)\mathfrak{b}_1 = (\Psi'_k\Psi_i\Psi_j/u)$, and because $\mathfrak{b}_k$ is a reduced principal ideal we must

have $\mathfrak{b}_k = \mathfrak{a}_m$ for some $m \in \mathbb{Z}_{>0}$. Furthermore,

$$\delta_m = \log(\Psi'_k \Psi_i \Psi_j / u) = \delta_i + \delta_j + \delta$$

where $\delta = \log(\Psi'_k/u)$. It can be shown that $\delta = O(\log D)$ which, when compared with δ_i and δ_j, is usually very small. Therefore, $\delta_m \approx \delta_i + \delta_j$. This almost gives us a group operation on the ideals in the cycle and enables us to reach ideals at fixed distances faster. As soon as we have an ideal $\mathfrak{a}_j$ with some large distance δ', we can use multiplications with $\mathfrak{a}_j$ followed by reduction instead of single applications of the continued fraction algorithm. This structure in the cycle of equivalent reduced ideals was called the *infrastructure* by its discoverer Shanks in [19].

3. (f, p) REPRESENTATIONS

In this section we introduce (f, p) representations, a technique for representing ideals and their distances in such a way that computations in the infrastructure can be done efficiently and with relatively low precision approximations of the distances. To keep track of distances when performing computations in the infrastructure, we would have to either store the distances with the ideals or their generators, as there is a one-to-one relationship between the distances and the generators of ideals. However, the coefficients of the generators grow exponentially in comparison to the distances of the ideals. On the other hand, maintaining accurate approximations of distances can be time-consuming.

The idea behind (f, p) representations, which were first introduced in Jacobson et al. [10] and improved in [11] by these same authors, is to store approximations of both the generator and the distance of an ideal. The approximations of generators are solely used for updating approximations of distances.

Using (f, p) representations has a number of important advantages over using other approaches for keeping track of distances. First of all, it is relatively easy to analyze the numerical accuracy of different kinds of operations on (f, p) representations. Furthermore, the precision that is required in order to obtain a certain level of accuracy tends to be lower than that required for other similar approaches. Finally, all operations on (f, p) representations involve only integer arithmetic, which makes algorithms that are implemented using (f, p) representations very fast compared to algorithms involving floating point operations.

We now give the definition of an (f, p) representation.

Definition 3.1. Let $p \in \mathbb{N}$, $f \in \mathbb{R}$ with $1 \leq f < 2^p$ and $\mathfrak{a}$ a primitive ideal. An (f, p) *representation* of $\mathfrak{a}$ is a triple $(\mathfrak{b}, d, k)$ where:

- $\mathfrak{b}$ is an ideal equivalent to $\mathfrak{a}$, $d \in \mathbb{N}$ with $2^p < d \leq 2^{p+1}$, $k \in \mathbb{Z}$,
- if $\theta \in \mathbb{Q}(\sqrt{\Delta})$ and $\mathfrak{b} = (\theta)\mathfrak{a}$, then $\left| \dfrac{2^{p-k}\theta}{d} - 1 \right| < \dfrac{f}{2^p}$.

We mainly look at the case when $\mathfrak{a} = (1)$. In the definition it is easy to see that when $\mathfrak{a} = (1)$, $2^{k-p}d$ is an approximation of the generator θ of $\mathfrak{b}$. Since $2^p < d \leq 2^{p+1}$, we can also see that $k \approx \log(2^{k-p}d) \approx \log\theta$, which makes k an approximation to the distance of the ideal $\mathfrak{b}$ (from $\mathfrak{a}$).

Performing computations with approximations gives rise to errors. From the definition we have that if a triplet $(\mathfrak{b}', d', k')$ is an (f, p) representation of $\mathfrak{a}$, then we have a relative error in storing the generator of $\mathfrak{b}$ (relative to $\mathfrak{a}$) that is smaller than $f/2^p$.

The following theorem, which can be found with proof in [11], shows how to compute products of (f,p) representations. Here we denote by $\lfloor y \rceil$ the nearest integer to $y \in \mathbb{R}$. This is the integer for which $-1/2 \le y - \lfloor y \rceil < 1/2$.

Theorem 3.2. *Let $(\mathfrak{b}', d', k')$ be an (f', p) representation of a primitive ideal $\mathfrak{a}'$ and $(\mathfrak{b}'', d'', k'')$ an (f'', p) representation of a primitive ideal $\mathfrak{a}''$. Put $d^* = \lfloor 2^{-p} d' d'' \rceil$, $d^{**} = \lfloor 2^{-(p+1)} d' d'' \rceil$, $k^* = k' + k''$, and*

$$(d,k) = \begin{cases} (d^*, k^*) & \text{if } d^* \le 2^{p+1}, \\ (d^{**}, k^* + 1) & \text{if } d^{**} \ge 2^p + 1, \\ (2^{p+1}, k^*) & \text{otherwise.} \end{cases}$$

Then $(\mathfrak{b}'\mathfrak{b}'', d, k)$ is an (f,p) representation of the primitive part of the product ideal $\mathfrak{a}'\mathfrak{a}''$ where $f = f^* + 1/2 + 2^{-(p+1)} f^*$ and $f^* = f' + f'' + 2^{-p} f' f''$.

In the cycle of ideals that we are interested in, all of the ideals are reduced. When we take the product of two reduced ideals, this product may not be reduced, so we have to apply the continued fraction algorithm to it in order to reduce it. Instead of first performing the multiplication and then the reduction we can use the algorithm NUCOMP in [11], which uses a more efficient method that almost simultaneously performs these two operations.

Remember that we work in the cycle of the reduced principal ideals, where we can consider distances modulo R_2 to be approximations of base 2-logarithms of generators of ideals. When we have an (f,p) representation of an ideal, it is clear that we immediately have k as an approximation of its distance. The following definition allows us, given such an approximation of a distance, to refer to an ideal close to this distance.

Definition 3.3. Let $\mathfrak{a}$ $(= \mathfrak{a}_1)$ be a given reduced ideal and let $(\mathfrak{a}_i,\ d_i,\ k_i)$ be a reduced (f,p)-representation of $\mathfrak{a}$. For $x \in \mathbb{Z}_{>0}$, we define $\mathfrak{a}(x)$ to be an ideal $\mathfrak{a}_j$ such that $k_j < x$ and $k_{j+1} \ge x$.

From here on we refer to an *ideal* $\mathfrak{a}(x)$, which represents an ideal $\mathfrak{a}_i$ with distance approximately x from the ideal (1), instead of the full (f,p) representation $(\mathfrak{a}_i, d_i, k_i)$ that is used to define it. Clearly, the definition is not very precise, in that it depends on k-values which are already approximations, and in that $\mathfrak{a}(x)$ yields an ideal with a k-value just smaller than x. So we need to know how close the distance of the ideal that we obtain is to x. In order to determine this, we require for the results in this paper that all (f,p) representations that are computed have $f < 2^{p-4}$. In [5] it is shown using techniques from [11] how p can be chosen to ensure that this is the case.

Using this requirement we obtain the following propositions, which are required for the results in the sequel. The first proposition can be used to determine how close the distance of an ideal $\mathfrak{a}(x)$ can be expected to be to x, while the second proposition shows us how to quickly obtain an ideal with a distance that is greater than or equal to x.

Proposition 3.4. *If $(\mathfrak{a}_i, d_i, k_i)$ is an (f,p) representation of $\mathfrak{a}_1 = (1)$ and $\mathfrak{a}_i = \mathfrak{a}(x)$, then if $f < 2^{p-4}$ we have*

- $\Psi_i < \frac{17}{8} 2^{k_i} \le \frac{17}{16} 2^x$,
- $\Psi_{i+1} > \frac{15}{16} 2^{k_{i+1}} \ge \frac{15}{16} 2^x$.

Proof. We have $x > k_i$, $k_{i+1} \geq x$,

$$\left| \frac{2^p \Psi_i}{d_i 2^{k_i}} - 1 \right| < \frac{f}{2^p} < \frac{1}{16}$$

and

$$\left| \frac{2^p \Psi_{i+1}}{d_{i+1} 2^{k_{i+1}}} - 1 \right| < \frac{1}{16} \ .$$

Hence,

$$\Psi_i < \frac{17}{16} \cdot \frac{d_i}{2^p} 2^{k_i} \leq 2 \cdot \frac{17}{16} 2^{k_i} \leq \frac{17}{8} 2^{x-1} = \frac{17}{16} 2^x$$

and

$$\Psi_{i+1} > \frac{15}{16} \cdot \frac{d_{i+1}}{2^p} 2^{k_{i+1}} > \frac{15}{16} 2^{k_{i+1}} \geq \frac{15}{16} 2^x. \qquad \square$$

Proposition 3.5. *For fixed p and f, if $\mathfrak{a}_i = \mathfrak{a}(x)$ and $\mathfrak{a}_j = \mathfrak{a}(x + \lambda)$, where $\lambda = \lceil \frac{1}{2} \log \Delta \rceil + 1$, then $j > i$.*

Proof. By Proposition 3.4, $\Psi_i < \frac{17}{16} 2^x$ and $\Psi_{j+1} > \frac{15}{16} 2^{x+\lambda} > \frac{15}{8} \sqrt{\Delta} 2^x$. Now $\Psi_{i+1} = \psi_i \Psi_i < \sqrt{\Delta} \frac{17}{16} 2^x$ and therefore $\Psi_{j+1} > \frac{15 \cdot 16}{8 \cdot 17} \Psi_{i+1} > \Psi_{i+1} \Rightarrow j > i$. $\square$

3.1. **Algorithms.** In this section we list the names and workings of the relevant algorithms that operate on (f, p) representations. To verify the regulator we only need the algorithms ADDX and AX, but since these are built up from other algorithms, we list these, too. Because the algorithms are already described in detail in [11] and/or [5], we only give a brief description here and refer to where they can be found.

The algorithm NUCOMP that we use, which is described extensively in [11], is the fastest known algorithm that both performs the multiplication of two ideals and the reduction of the result. On input of a reduced (f', p) representation $(\mathfrak{b}', d', k')$ of a primitive ideal $\mathfrak{a}'$ and a reduced (f'', p) representation $(\mathfrak{b}'', d'', k'')$ of a primitive ideal $\mathfrak{a}''$, it outputs a reduced (f, p) representation $(\mathfrak{b}, d, k)$ of $\mathfrak{a}'\mathfrak{a}''$ where $f = f^* + 13/8 + 2^{-(p+1)} f^*$ with $f^* = f' + f'' + 2^{-p} f' f''$. The version of NUCOMP that is used here differs from the original version that was originally introduced by Shanks [20] in that it operates on (f, p) representations rather than binary quadratic forms.

The next algorithm, ADJUST, enables us to skip to an ideal $\mathfrak{a}(x)$ through the cycle of reduced principal ideals starting at another ideal, provided that the third parameter k in the (f, p) representation of this ideal is such that $k < x$. On input of $x \in \mathbb{Z}_{>0}$ and a reduced (f, p) representation $(\mathfrak{b}, d, k)$ of a primitive ideal $\mathfrak{a}$ with $k < x$, it outputs a reduced $(f + 9/8, p)$ representation $(\mathfrak{a}(x), g, h)$ of $\mathfrak{a}$. It works by applying the continued fraction algorithm (by taking one step through the cycle at a time), but is more efficient because, like NUCOMP, it applies the algorithm to a rational approximation of the irrational $(P + \sqrt{D})/Q$, which gives the same outcome for the required number of iterations of this algorithm. The algorithm, which can be found in [5], is a slightly modified version of the algorithm NEAR that can be found in [11].

Briefly put, the algorithm ADDX, which can be found in [5], gives us the ability to multiply two ideals $\mathfrak{a}(x)$ and $\mathfrak{a}(y)$ and thereby obtain the ideal $\mathfrak{a}(x + y)$. More specifically, on input of a reduced (f', p) representation $(\mathfrak{a}(x'), d', k')$ of the ideal $\mathfrak{a} = (1)$ and a reduced (f'', p) representation $(\mathfrak{a}(x''), d'', k'')$ of $\mathfrak{a}$, it outputs a reduced (f, p) representation $(\mathfrak{a}(x' + x''), d, k)$ of $\mathfrak{a}$, where $f = f^* + 11/4 + 2^{-(p+1)} f^*$

and $f^* = f' + f'' + 2^{-p} f' f''$. This enables us to jump quickly through the cycle using infrastructure.

Finally, we have the algorithm AX which, given a distance $x \in \mathbb{Z}_{>0}$, outputs a certain (f, p) representation $(\mathfrak{a}(x), d, k)$ of $\mathfrak{a} = (1)$. Essentially, the algorithm performs simple binary exponentiation for ideals and updates the corresponding (f, p) representations. A complete description of this algorithm can also be found in [5].

4. Regulator verification

As much of the work described here is intended for computer implementation, it is more convenient to work with R_2 instead of R. In the remainder of this paper we therefore assume that we have been provided with a value R_2' that is a sufficiently close approximation to a multiple of R_2, i.e., a value R_2' for which $|R_2' - cR_2| < 1$ for some $c \in \mathbb{Z}_{>0}$. Such a value can be obtained using a subexponential method, possibly after refining the original outcome (see Algorithm 5.10 in [5]). The goal of the verification is to verify that $c = 1$ or, equivalently, that $|R_2' - R_2| < 1$.

As described in the introduction, the verification procedure consists of two main parts. First, it uses a baby step/giant step technique to determine a lower bound for R_2. Then it uses a different technique to determine the multiplier in R_2', i.e., the value c so that R_2' is an approximation to cR_2. For the latter technique we need to check all possible prime divisors of c, but this collection of primes is limited because of the lower bound that has been determined using the baby step/giant step technique. The next two sections provide the mathematical results on which these techniques are based.

4.1. Baby steps and giant steps.

In Section 2.1.1 we described how we can compute the fundamental unit and therefore, by looking at distances, compute the regulator by computing all of the ideals in the cycle of reduced principal ideals using the continued fraction algorithm. Furthermore, in Section 2.2 we showed how we can skip steps of the algorithm using a combination of multiplication and reduction. A single application of the continued fraction algorithm is also referred to as a *baby step* and the combination of a multiplication and a reduction is also referred to as a *giant step*.

Our version of the baby step/giant step technique is based on Theorem 4.3, which states that if we store a list with a certain number of ideals (a *baby step list*) from the starting ideal, where this number depends on the size of the giant steps, then one of the ideals computed during the giant steps or one of their conjugates has to end up in this baby step list after a certain number of iterations if $\lceil R_2 \rceil \leq K$ for some given K. The reason that we can also consider conjugates has to do with the symmetry property that was described at the end of Section 2.1.1.

In order to prove Theorem 4.3 we require two preliminary results, both of which partially determine the balance between the size of the baby step list and the size of the giant steps that is required in order to end up in the baby step list after completing the cycle of reduced principal ideals. Theorem 4.1 determines this balance in the situation where we look at the ideal that we get after taking a giant step, and Theorem 4.2 determines this balance when looking at the conjugate of this ideal.

Theorem 4.1. *Let $x = \lceil cR_2 \rceil$, where $c \in \mathbb{Z}_{>0}$, and let $(\mathfrak{a}_j, d_j, k_j)$ and $(\mathfrak{a}_i, d_i, k_i)$ be (f,p) representations of $\mathfrak{a}_1 = (1)$ such that $\mathfrak{a}_j = \mathfrak{a}(x+r)$ and $\mathfrak{a}_i = \mathfrak{a}(r)$ with $r \in \mathbb{Z}_{>0}$. Then $\mathfrak{a}(x+r) \in \{\mathfrak{a}_{\max\{1,i-2\}}, \ldots, \mathfrak{a}_{i+3}\}$.*

Proof. We have $k_j < x + r$, $b_{j+1} \geq x + r$, $x = \log \eta + \epsilon$ for $\eta = (2^{R_2})^c$ and some ϵ with $0 < \epsilon < 1$. By Proposition 3.4, we have $\Psi_j < \frac{17}{16} 2^{x+r} = \eta \frac{17}{8} 2^{\epsilon + r - 1}$ and $\Psi_{j+1} > \frac{15}{16} 2^{x+r} = \eta \frac{15}{16} 2^{\epsilon + r} > 1$ (as $r \geq 1$ and $\eta > 1$).

Put $\Psi_m = \eta^{-1} \Psi_{j+1}$. Then also $\Psi_m > 1$, so we must have $m > 1$, which implies that $m - 1 \geq 1$. Note that by Proposition 3.4, $\Psi_{m-1} = \eta^{-1} \Psi_j < \frac{17}{8} 2^{\epsilon + r - 1}$ and $\Psi_m > \frac{15}{16} 2^{\epsilon + r}$, and similarly we have $\Psi_i < \frac{17}{16} 2^r$ and $\Psi_{i+1} > \frac{15}{16} 2^r = \frac{15}{8} 2^{r-1}$.

Now $\Psi_m > \frac{15}{17} 2^{\epsilon} \Psi_i > \frac{15}{17} \Psi_i \Rightarrow \Psi_{m+2} > F_3 \Psi_m = 2\Psi_m > \frac{2 \cdot 15}{17} \Psi_i > \Psi_i \Rightarrow m + 1 \geq i \Rightarrow m - 1 \geq i - 2$. Also, $\Psi_{m-1} < \frac{17}{15} 2^{\epsilon} \Psi_{i+1} < 3\Psi_{i+1} = F_4 \Psi_{i+1} < \Psi_{i+4} \Rightarrow m - 1 < i + 4 \Rightarrow m - 1 \leq i + 3$.

By the definition of Ψ_m, we have $\mathfrak{a}_{m-1} = (\eta^{-1} \Psi_j) \mathfrak{a}_1 = (\Psi_j) \mathfrak{a}_1 = \mathfrak{a}_j$ which, together with our determined bounds on $m - 1$, gives us the required result. □

Theorem 4.2. *Put $\lambda = \lceil \frac{1}{2} \log \Delta \rceil + 1$. Let $r \in \mathbb{Z}_{<0}$, $x = \lceil cR_2 \rceil$ and $x + r > 0$. Put $\mathfrak{a}_j = \mathfrak{a}(x+r) \neq \mathfrak{a}_1$, $\mathfrak{a}_i = \mathfrak{a}(|r|)$, $\mathfrak{a}_t = \mathfrak{a}(|r| + \lambda)$ and assume that $f < 2^{p-4}$. Then $\overline{\mathfrak{a}}(x+r) \in \{\mathfrak{a}_{\max\{2,i-1\}}, \ldots, \mathfrak{a}_t\}$.*

Proof. Again we have that $x = \log \eta + \epsilon$, for $\eta = (2^{R_2})^c$ and some ϵ with $0 < \epsilon < 1$. By Proposition 3.4 we have $\Psi_j < \frac{17}{16} 2^{x+r} = \eta \frac{17}{16} 2^{\epsilon + r}$, from which we can deduce that $\Psi_j |\overline{\Psi}_j| = N(\mathfrak{a}_j) \Rightarrow |\overline{\Psi}_j| > N(\mathfrak{a}_j) \eta^{-1} \frac{16}{17} 2^{|r| - \epsilon} \Rightarrow \eta |\overline{\Psi}_j| > N(\mathfrak{a}_j) \frac{16}{17} 2^{|r| + \epsilon} > 1$, because $N(\mathfrak{a}_j) > 1$ and $|r| \geq 1$.

Since $\overline{\mathfrak{a}}_j$ is reduced and $\neq \mathfrak{a}_1$, $\overline{\mathfrak{a}}_j = \mathfrak{a}_m$ for some m such that $m > 1$ and $\Psi_m = \eta |\overline{\Psi}_j| > 1$. From this it follows that $\eta |\overline{\Psi}_{j+1}| = \Psi_{m-1}$ because $\overline{\mathfrak{a}}_{j+1} = \mathfrak{a}_{m-1}$. Now by Proposition 3.4 and Theorem 2.1, $\Psi_i < \frac{17}{16} 2^{|r|} < (\frac{17}{16})^2 2^{\epsilon} N(\mathfrak{a}_j)^{-1} \eta |\overline{\Psi}_j| = (\frac{17}{16})^2 2^{\epsilon} N(\mathfrak{a}_j)^{-1} \Psi_m < 2\Psi_m = F_3 \Psi_m < \Psi_{m+2}$. So $i < m + 2 \Rightarrow m \geq i - 1$ and we already had $m > 1$.

Also, again using Proposition 3.4, $\Psi_{j+1} = \Psi_j \psi_j \Rightarrow \Psi_j = \frac{1}{\psi_j} \Psi_{j+1} > \frac{1}{\psi_j} \frac{15}{16} 2^{x+r} = \frac{1}{\psi_j} \frac{15}{16} \eta 2^{r+\epsilon} \Rightarrow |\overline{\Psi}_j| < \psi_j \frac{16}{15} \eta^{-1} 2^{|r| - \epsilon} N(\mathfrak{a}_j)$, and Section 2.1.1 tells us that $\psi_j N(\mathfrak{a}_j) < \sqrt{\Delta}$, so that $\Psi_m = \eta |\overline{\Psi}_j| < \sqrt{\Delta} 2^{|r| - \epsilon} \frac{16}{15}$. Now $\Psi_{t+1} > \frac{15}{16} 2^{|r| + \lambda} > \frac{30}{16} \sqrt{\Delta} 2^{|r|} > \frac{30}{16} \cdot \frac{15}{16} 2^{\epsilon} \Psi_m > \Psi_m$ and $t + 1 > m \Rightarrow m \leq t$. □

With these results we can now prove the main theorem, which is required for the baby step/giant step algorithm that is used for determining a lower bound for $\lceil R_2 \rceil$. Theorem 4.3 states that we can define our baby step list $\mathcal{L}$ to be the list $\{\mathfrak{a}_1, \mathfrak{a}_2, \ldots, \mathfrak{a}_{t+2}\}$, where $\mathfrak{a}_t = \mathfrak{a}(s + \lambda)$ for some $s \in \mathbb{Z}_{>0}$, after which we can take giant steps of size $2s$. Here $\lambda = \lceil \frac{1}{2} \log \Delta \rceil + 1$.

Theorem 4.3. *Let $x = \lceil cR_2 \rceil$, $c \in \mathbb{Z}_{>0}$ and suppose that $x = 2qs - r$ (for $0 < |r| \leq s$, $s > 1$ and $r, s \in \mathbb{Z}$). Then $\mathfrak{a}(2qs)$ or $\overline{\mathfrak{a}}(2qs) \in \{\mathfrak{a}_1, \mathfrak{a}_2, \ldots, \mathfrak{a}_{t+2}\}$, where $\mathfrak{a}_t = \mathfrak{a}(s + \lambda)$.*

Proof. Assume that $\mathfrak{a}_l = \mathfrak{a}(s)$ and $\mathfrak{a}_i = \mathfrak{a}(|r|)$. We distinguish between when $r > 0$ and when $r < 0$, but in both cases we have $i \leq l < t$ by Proposition 3.5 and the fact that $|r| \leq s$.

- If $r > 0$, we have $\mathfrak{a}_i = \mathfrak{a}(r)$, so that $\mathfrak{a}(x + r) \in \{\mathfrak{a}_{\max\{1,i-2\}}, \ldots, \mathfrak{a}_{i+3}\}$ by Theorem 4.1. Furthermore, $\{\mathfrak{a}_{\max\{1,i-2\}}, \ldots, \mathfrak{a}_{i+3}\} \subseteq \{\mathfrak{a}_1, \ldots, \mathfrak{a}_{t+2}\}$, because $i < t$.

- If $r < 0$, then either $\overline{\mathfrak{a}}(x+r) = \mathfrak{a}_1$ or $\overline{\mathfrak{a}}(x+r) \neq \mathfrak{a}_1$. If $\overline{\mathfrak{a}}(x+r) \neq \mathfrak{a}_1$, then $\overline{\mathfrak{a}}(x+r) \in \{\mathfrak{a}_{\max\{2,i-1\}}, \ldots, \mathfrak{a}_v\}$ for some $v \leq t$ by Theorem 4.2 and the fact that $|r| \leq s$. So either way, $\overline{\mathfrak{a}}(x+r) \in \{\mathfrak{a}_1, \mathfrak{a}_{\max\{2,i-1\}}, \ldots, \mathfrak{a}_v\} \subseteq \{\mathfrak{a}_1, \ldots, \mathfrak{a}_{t+2}\}$.

Because $\mathfrak{a}(x+r) = \mathfrak{a}(2qs)$ this concludes our proof. □

If $\mathfrak{a}(2qs)$ and $\overline{\mathfrak{a}}(2qs)$ are not in the list $\{\mathfrak{a}_1, \mathfrak{a}_2, \ldots, \mathfrak{a}_{t+2}\}$ for $q = 1, 2, \ldots, Q$, then $\lceil cR_2 \rceil > 2Qs + s > 2Qs$ for all $c \in \mathbb{Z}_{>0}$, i.e., $\lceil R_2 \rceil > 2Qs$. Therefore, if we choose s and Q large enough we can determine a lower bound K for $\lceil R_2 \rceil$ by verifying membership of $\mathfrak{a}(2qs)$ and $\overline{\mathfrak{a}}(2qs)$ for $q = 1, \ldots, Q$. The algorithm that builds the baby step list and performs these membership tests is described in more detail in [5].

4.2. **Determining the multiplication factor.** When we have a value R_2' such that $|R_2' - cR_2| < 1$ for some $c \in \mathbb{Z}_{>0}$, there is an ideal at or near distance R_2' that is equal to $\mathfrak{a}_1$. We only get ideals equal to $\mathfrak{a}_1$ after precisely traversing a number of cycles of reduced principal ideals starting at $\mathfrak{a}_1$, which means that if an ideal at distance near $d = R_2'/q$ with $q \in \mathbb{Z}_{>0}$ is equal to $\mathfrak{a}_1$, then d must be equal or very close to tR_2 for some $t \in \mathbb{Z}_{>0}$ where $t \mid c$ ($c \approx R_2'/R_2$). In other words, we can determine the multiplication factor of R_2' by dividing by integers and checking whether we can find an ideal near the resulting distance that is equal to $\mathfrak{a}_1$.

Of course we want to make sure that we do not count any possible factors of R_2'/R_2 more than once. We can do this by continuing with the smaller multiple R_2'/q after finding out that an ideal close enough to distance R_2'/q is equal to $\mathfrak{a}_1$. This way we get rid of the factors we already determined and cannot find them more often then they occur. Naturally, it is best to let the tested values for q range over the primes.

It is important to first investigate how close to $\mathfrak{a}_1$ we end up when a distance is close enough to a multiple of R_2 and whether we can also get that close to $\mathfrak{a}_1$ when it is not. This way, we can precisely specify when we consider a distance R_2'/q to be close enough to the ideal $\mathfrak{a}_1$ to decide that q is a factor of R_2'/R_2.

Theorem 4.4. *Let $y = \lceil \log \Psi_j \rceil + \gamma$, where $\gamma \in \{-1, 0, 1\}$. If $\mathfrak{a}_i = \mathfrak{a}(y)$ and $y > 1$, then* $\max\{1, j-3\} \leq i \leq j+3$.

Proof. We first note that $0 \leq y - 2 < \log \Psi_j < y + 1$. We have $\mathfrak{a}(y) = (\Psi_i)\mathfrak{a}_1$, $\Psi_i < \frac{17}{16}2^y$ and $\Psi_{i+1} > \frac{15}{16}2^y$ by Proposition 3.4. Furthermore, by Theorem 2.1, $\Psi_{i+4} > F_4\Psi_{i+1} = 3\Psi_{i+1} > \frac{3 \cdot 15}{16}2^y > 2^{y+1} \geq \Psi_j$. Hence, $i+4 > j \Rightarrow i+3 \geq j$. Also, $F_5\Psi_{i-4} < \Psi_i < \frac{17}{16}2^y = \frac{17}{4}2^{y-2} \leq \frac{17}{4}\Psi_j \Rightarrow \Psi_{i-4} < \Psi_j \Rightarrow i-4 < j \Rightarrow i-3 \leq j$ and because $y > 1$ and $\Psi_1 = 1$ we must have $i \geq 1$. □

Corollary 4.5. *If $\mathfrak{a}_1 = (1)$ and η ($> \Delta^{3/2}$) is any unit of $\mathbb{Q}(\sqrt{D})$, then if $x = \lceil \log \eta \rceil + \gamma \geq 2$ ($\gamma \in \{-1, 0, 1\}$), we must have that $\mathfrak{a}(x) \in \{\overline{\mathfrak{a}}_4, \overline{\mathfrak{a}}_3, \overline{\mathfrak{a}}_2, \mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{a}_3, \mathfrak{a}_4\}$.*

Proof. We know that $\mathfrak{a}_j = (\eta)\mathfrak{a}_1$, where $\eta = \Psi_j$. Hence, by Theorem 4.4, $\mathfrak{a}(x) \in \{\mathfrak{a}_{j-3}, \mathfrak{a}_{j-2}, \mathfrak{a}_{j-1}, \mathfrak{a}_j, \mathfrak{a}_{j+1}, \mathfrak{a}_{j+2}, \mathfrak{a}_{j+3}\}$. We have $\Psi_j < (\sqrt{\Delta})^3\Psi_{j-3} \Rightarrow \Psi_{j-3} > 1 \Rightarrow j - 3 > 1$, so $j > 3$ and furthermore $\mathfrak{a}_j = (\Psi_j)\mathfrak{a}_1 = (\eta)\mathfrak{a}_1 = \mathfrak{a}_1 \Rightarrow \mathfrak{a}_{j-3} = \overline{\mathfrak{a}}_4$, $\mathfrak{a}_{j-2} = \overline{\mathfrak{a}}_3$, etc. □

The corollary above tells us that when we do not end up in one of the above-mentioned seven ideals at approximate distance $\lceil R_2'/q \rceil$, our prime q is not a factor of the multiplier c, where $|R_2' - cR_2| < 1$. We would like this set of seven ideals

to be exclusive, so that finding one of these ideals assures us that we have indeed found a prime factor.

It appears that the converse of the corollary is only true once we impose some restriction on the size of the primes we divide out. Intuitively, this makes sense, as dividing out by a prime that is approximately equal to R_2' is very likely to give us an ideal with a small enough distance to be in the list, even though this prime does not give us a factor of the multiplier.

Lemma 4.6. *If $x = y + \gamma$, where $x, y \in \mathbb{Z}$ and $\gamma \in \{-1, 0, 1\}$, then $\lceil x/q \rceil = \lceil y/q \rceil + \gamma'$ where $\gamma' \in \{-1, 0, 1\}$, for any $q \geq 1$.*

Theorem 4.7. *Put $x = \lceil \frac{\log \eta + \gamma}{q} \rceil$ where $\gamma \in \{-1, 0, 1\}$, $\eta = {\eta_0}^m$ ($m \in \mathbb{Z}$) and η_0 is the fundamental unit of $\mathbb{Q}(\sqrt{D})$. If $R_2 = \log \eta_0 > q \cdot (2 \log \Delta + \log \frac{17}{4})$, we must have $q \mid m$ when $\mathfrak{a}(x) \in \{\overline{\mathfrak{a}}_4, \overline{\mathfrak{a}}_3, \overline{\mathfrak{a}}_2, \mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{a}_3, \mathfrak{a}_4\}$.*

Proof. Let $\mathfrak{a}_j = \mathfrak{a}(x)$. If we do have $\mathfrak{a}_j = \mathfrak{a}_i$ or $\mathfrak{a}_j = \overline{\mathfrak{a}}_i$ with $i \leq 4$, then either $\Psi_j/\Psi_i = {\eta_0}^k$ or $\Psi_j/|\Psi_i| = {\eta_0}^k$ for some $i \leq 4$ and $k < m$.

Using Proposition 3.4 and Lemma 4.6, we can deduce that

$$\Psi_j < \frac{17}{16} 2^{\lceil \frac{\log \eta + \gamma}{q} \rceil} = \frac{17}{16} 2^{\lceil \frac{\log \eta}{q} \rceil + \gamma'} = \frac{17}{16} {\eta_0}^{\frac{m}{q}} 2^{\epsilon + \gamma'} \tag{4.1}$$

and

$$\Psi_{j+1} > \frac{15}{16} 2^{\lceil \frac{\log \eta + \gamma}{q} \rceil} = \frac{15}{16} 2^{\lceil \frac{\log \eta}{q} \rceil + \gamma'} = \frac{15}{16} {\eta_0}^{\frac{m}{q}} 2^{\epsilon + \gamma'}$$

$$\Rightarrow \Psi_j > \frac{1}{\psi_j} \frac{15}{16} {\eta_0}^{\frac{m}{q}} 2^{\epsilon + \gamma'} \tag{4.2}$$

for some $\gamma' \in \{-1, 0, 1\}$ and some ϵ with $0 < \epsilon < 1$.

Now put $\nu = \Psi_i$ if $\mathfrak{a}_j = \mathfrak{a}_i$ or $\nu = |\overline{\Psi}_i|$ if $\mathfrak{a}_j = \overline{\mathfrak{a}}_i$, so that $\Psi_j = \nu {\eta_0}^k$. If $\nu = \Psi_i$, then $\nu \leq \Psi_4 < (\sqrt{\Delta})^3$ as $i \leq 4$. If $\nu = |\overline{\Psi}_i|$, then since $\Psi_i |\overline{\Psi}_i| = N(\mathfrak{a}_i)$ and $i \leq 4$ we get $\frac{1}{\nu} \geq \frac{1}{\Psi_4} > \frac{1}{(\sqrt{\Delta})^3}$. Hence, $|\log \nu| < 3 \log \sqrt{\Delta} = \frac{3}{2} \log \Delta$.

Using (4.1) we obtain

$$\nu {\eta_0}^k = \Psi_j < \frac{17}{16} 4 {\eta_0}^{m/q}$$

$$\Rightarrow \log \nu + k R_2 < \log \frac{17}{4} + \frac{m}{q} R_2,$$

and using (4.2) we get

$$\nu {\eta_0}^k = \Psi_j > \frac{1}{\sqrt{\Delta}} \frac{15}{16} {\eta_0}^{m/q} \frac{1}{2} = \frac{1}{\sqrt{\Delta}} \frac{15}{32} {\eta_0}^{m/q}$$

$$\Rightarrow \log \nu + k R_2 > -\log \sqrt{\Delta} + \log \frac{15}{32} + \frac{m}{q} R_2 \ .$$

Hence,

$$-\log \nu - \log \sqrt{\Delta} + \log \frac{15}{32} < \left(k - \frac{m}{q} \right) R_2 < -\log \nu + \log \frac{17}{4},$$

$$\left| \left(k - \frac{m}{q} \right) R_2 \right| < 2 \log \Delta + \log \frac{17}{4},$$

$$|(kq - m)| \, R_2 < q \cdot \left(2 \log \Delta + \log \frac{17}{4} \right).$$

So when $R_2 > q \cdot (2\log\Delta + \log\frac{17}{4})$, we must have $(kq - m) = 0 \Rightarrow m = kq \Rightarrow q \mid m$. □

It is clear that we can limit ourselves to testing only those primes below R_2', but this still involves an unnecessarily large number of checks. This is where the determination of the lower bound during the baby step/giant step algorithm comes in. If we have verified lower bound K, we know that $c \approx R_2'/R_2 < R_2'/K$. Because of this, we need only check all primes below R_2'/K. In order to ensure that $c < R_2'/\delta$ where $\delta = 2\log\Delta + \log\frac{17}{4}$ (as is required by Theorem 4.7 when c is prime) we now merely have to make sure that $K > \sqrt{R_2'\delta}$, as then $c \approx R_2'/R_2 < R_2'/K = R_2'K/K^2 < R_2'K/(R_2'\delta) = K/\delta < R_2/\delta$.

4.3. **Complexity results.** We want to select a bound K for the verification algorithm that gives us an optimal run time complexity. In order to do this we have to select K such that the two main parts of the algorithm have the same complexity, as otherwise the half with the largest complexity dominates the other half. We start the analysis by looking at the run time complexities of the separate halves of the algorithm. In the discussion we take the time needed for computing a baby step as a time unit and use the fact that the ratio between the time for computing a giant step as opposed to a baby step is asymptotic to $\log D$. We also assume that the time required for searching a table is constant.

4.3.1. *Complexity of the baby step/giant step algorithm.* The baby step/giant step algorithm consists of the creation of the baby step list (BS) and computing the giant steps (GS). During the baby step phase (BS) we compute all of the baby steps up to distance $s + \lambda$ plus an additional two. Because of equation (2.2), we expect the number of baby steps that are computed during the baby step phase to be proportional in size to $s + \lambda$. This gives us run time complexity $O(s)$ for (BS).

During the giant step phase (GS) we perform Q giant steps, which gives us run time complexity of $O(Q)$ baby steps. Since we require that $2sQ > K$, we see that we get the best run time complexity for algorithm (BS+GS) when $s = Q = \sqrt{K}$. Thus, the total complexity of the algorithm (BS+GS) is $O(K^{1/2} + K^{1/2+\epsilon}) = O(K^{1/2+\epsilon})$ baby steps.

4.3.2. *Complexity of determining the multiplier.* The running time for determining the value of c (FM) mainly depends on the time needed to locate the ideals at the given distances, as the times needed to find the next prime and to check whether the ideal found is in the list are extremely small by comparison.

Using the algorithm AX we can compute an ideal at a certain distance X using $O(\log X)$ ideal multiplications that are each followed by a reduction step. The largest distance that we have to check is $\lceil R_2'/2 \rceil$. Using the prime number theorem, that states that $\lim_{x\to\infty} \pi(x)/\frac{x}{\ln x} = 1$, where $\pi(x)$ stands for the number of primes below x, we find that $M/\ln M$ gives us an estimate of the number of possible prime divisors below M. Because of the predetermined lower bound K for R_2, we can use $M = R_2'/K$.

Assuming the worst-case scenario $R_2' = R_2$, where neither the search space nor the distances to check is reduced during execution of the algorithm (the upper bound for the primes becomes M/q and R_2' becomes R_2'/q after finding a prime factor q), we can approximate the running time of (FM) by $O(\frac{M}{\ln M}\log(R_2'/2)) =$

$O(\frac{M}{\log M}\log R_2') = O(\frac{R_2'/K}{\log(R_2'/K)}\log R_2') = O(\frac{R_2'}{K}\frac{\log R_2'}{\log(R_2'/K)})$ giant steps, or equivalently $O(\frac{R_2'}{K}\frac{\log R_2'\log D}{\log(R_2'/K)})$ baby steps.

4.3.3. *Optimal complexity for the verification algorithm.* Now that we have the run time complexities of the two halves of the algorithm, we can determine the optimal complexity for the entire algorithm. We know that the total running time can be found by adding up the time for (BS+GS) and (FM), which is

$$O\left(K^{1/2+\epsilon} + \frac{R_2'}{K}\frac{\log R_2'\log D}{\log(R_2'/K)}\right),$$

so we want K to be optimal for this formula.

To make the complexities of the two halves about equal, we want $\sqrt{K} = R_2'/K$ (ignoring log factors), which tells us that we should choose $K = R_2'^{\frac{2}{3}}$. Thus, the optimal running time for the entire algorithm ((BS+GS)+FM) is

$$O\left((R_2'^{\frac{2}{3}})^{1/2+\epsilon} + \frac{R_2'}{R_2'^{\frac{2}{3}}}\frac{\log R_2'\log D}{\log\left(R_2'/R_2'^{\frac{2}{3}}\right)}\right) = O\left(R_2'^{\frac{1}{3}+\epsilon} + R_2'^{\frac{1}{3}+\epsilon}\right) = O\left(R_2'^{\frac{1}{3}+\epsilon}\right) \quad .$$

Now we look at the situation where we have combined the verification algorithm with a subexponential algorithm, so that we get an algorithm that unconditionally computes R_2, and determine the complexity. Under assumption of the GRH, the approximation R_2' is close to R_2 instead of close to a multiple of R_2, and we expect R_2 to be approximately of size $\sqrt{D}$. Furthermore, under assumption of the GRH, the subexponential algorithms have expected subexponential run time complexity. Therefore, it is easy to see that under assumption of the GRH we have expected run time complexity $O(\sqrt{D}^{1/3+\epsilon}) = O(D^{1/6+\epsilon})$ for the combination of a subexponential algorithm and the verification algorithm.

5. Practical improvements

5.1. **Memory issues.** In practice, we only have a limited amount of storage space available to store the baby step list $\mathcal{L}$. Since we have a fixed set of ideals to compute for the baby step list (all ideals up to distance $s+\lambda$ and two beyond this distance), we have to somehow limit the number of ideals that we are going to store, which we do by introducing gaps in the baby step list. Note that equation (2.2) shows us there are roughly $t = (s+\lambda)/1.7$ ideals with distance smaller than $s+\lambda$.

Assume that $\mathfrak{a}_u = \mathfrak{a}(s+\lambda)$. Furthermore, assume that we have space for storing $N+4$ ideals and that our approximation t of u (the number of ideals below distance $s+\lambda$) is such that $u < 1.05t$.[1] When $1.05t \le N$, we have enough space to store the entire baby step list $\mathcal{L}$. However, as soon as $1.05t > N$, we need to leave out part of the baby step list. We represent the ratio of the ideals below bound $s+\lambda$ for which we have enough space using the variable $l \in \mathbb{Z}_{>0}$, where l is the smallest positive integer for which we have that $1.05t/l \le N$. Equivalently, $l = \lceil 1.05t/N \rceil$. Instead of storing the entire list $\mathcal{L}$, we store the sublist

$$\mathcal{L}' = \{\mathfrak{a}_1, \mathfrak{a}_l, \mathfrak{a}_{2l}, \ldots, \mathfrak{a}_u = \mathfrak{a}(s+\lambda), \mathfrak{a}_{u+1}, \mathfrak{a}_{u+2}\},$$

[1] During our experiments, t turned out to be slightly larger than u already, which is most likely (see Section 2.2) because $\tau \log e \approx 1.711\ldots$, which is slightly larger than the value 1.7 that we use.

which contains every l^{th} ideal before ideal $\mathfrak{a}_u$ and the ideals $\mathfrak{a}_1$, $\mathfrak{a}_u$, $\mathfrak{a}_{u+1}$ and $\mathfrak{a}_{u+2}$. Because of the way in which we determined l, we know that we have enough space to store these ideals. It is now fairly easy to show that if we have an ideal $\mathfrak{a}_j$ that is in $\mathcal{L}$, then at least one of the ideals in the list $\mathcal{N} := \{\mathfrak{a}_j, \mathfrak{a}_{j+1}, \ldots, \mathfrak{a}_{j+l-1}\}$ has to be in $\mathcal{L}'$, so that we can replace matching of the ideal $\mathfrak{a}_j$ with the ideals in the list $\mathcal{L}$ with an iteration that tries to match the ideals in $\mathcal{N}$ with the ideals in $\mathcal{L}'$. The details for using this method with our verification method can be found in [5].

5.2. **Improving the algorithm for determining the multiplier.** We present an improvement, first introduced by Jacobson et al. in [8], to the algorithm for determining the multiplier. In our presentation, the original method has been modified to work with (f, p) representations and is described in more detail than in [8]. Furthermore, we accompany the description of this improvement by a theorem that ensures the correctness of its use with our verification algorithm.

While trying to determine the multiplier c, we repeatedly have to compute an ideal $\mathfrak{a}(\lceil R_2'/q_i \rceil)$ and determine whether this ideal is in the list from Theorem 4.7. We start this procedure with the largest prime and proceed with the primes sorted in a decreasing order, so that the distances R_2'/q_j increase during the algorithm. Instead of computing the ideal $\mathfrak{a}(\lceil R_2'/q_i \rceil)$ from scratch for every prime q_i, we can now use the fact that we have already computed the ideal $\mathfrak{a}(\lceil R_2'/q_{i-1} \rceil)$ in the preceding step (where $q_i < q_{i-1}$), which lies close to the new distance $\lceil R_2'/q_i \rceil$. Because we already have this ideal, we can put $\delta' := \lceil R_2'/q_{i-1} \rceil$, determine the difference $\delta := \lceil R_2'/q_i \rceil - \delta'$ between the distances of the ideals and use the infrastructure to take a giant step from $\mathfrak{a}(\lceil R_2'/q_{i-1} \rceil)$ to $\mathfrak{a}(\lceil R_2'/q_i \rceil)$ using the ideal $\mathfrak{a}(\delta)$ and the algorithm ADDX.

To make the process of computing the ideals $\mathfrak{a}(\delta)$ more efficient, we can further optimize the algorithm by approximating these ideals using products of precomputed ideals. We first precompute all ideals $\mathfrak{a}_{t_0}, \mathfrak{a}_{t_1}, \ldots, \mathfrak{a}_{t_m}$ where $\mathfrak{a}_{t_i} = \mathfrak{a}(\delta_{t_i})$, $\delta_{t_i} = 2^i(s')$, $s' = s - 1$ and m depends on which ideals we expect to use. Now, after computing an ideal at distance $\delta' \approx \lceil R_2'/q_{i-1} \rceil$, we compute δ and put $\rho := \lfloor \delta/s' \rfloor + 1$, so that $\delta < \rho s' \leq \delta + s'$. We can use the binary expansion $b_r 2^r + b_{r-1}2^{r-1} + \cdots + b_0$ of ρ to get $\delta \approx \rho s' = \rho\delta_{t_0} = 2^r\delta_{t_0} + b_{r-1}2^{r-1}\delta_{t_0} + \cdots + b_0\delta_{t_0} = b_r\delta_{t_r} + b_{r-1}\delta_{t_{r-1}} + \cdots + b_0\delta_{t_0}$. So if $\mathfrak{a}_{s_{q_{i-1}}} = \mathfrak{a}(\delta')$, where $\delta' \approx \lceil R_2'/q_{i-1} \rceil$, we can find an ideal at approximate distance $\lceil R_2'/q_i \rceil$ by computing the ideal $\mathfrak{a}(\delta' + \rho s')$. Here we can easily compute ideal $\mathfrak{a}(\rho s')$ by applying ADDX to the ideals $\mathfrak{a}_{t_i}$ for which $b_i = 1$, after which we can use $\mathfrak{a}(\rho s')$ and $\mathfrak{a}(\delta')$ to efficiently compute the ideal $\mathfrak{a}(\delta' + \rho s')$.

Because we actually add $\rho s'$ to the distance δ' of our ideal $\mathfrak{a}_{s_{q_{i-1}}}$ instead of δ, we do not in general compute the ideal $\mathfrak{a}(\lceil R_2'/q_1 \rceil)$ but obtain an ideal that is close by. The relevance of the next theorem is that it shows us that we can use the technique described above to quickly compute an ideal near distance $\lceil R_2'/q \rceil$ for each prime q and know that the q does not divide the multiplier c for which $|R_2' - cR_2| < 1$ when the ideal that we compute does not end up in the list mentioned in the theorem. As a result, we only need to compute the precise ideals $\mathfrak{a}(\lceil R_2'/q_j \rceil)$ using AX to verify that $q \nmid c$, when our approximated ideal ends up in the list.

Theorem 5.1. *Suppose that c is a positive integer and $|R_2' - cR_2| < 1$. Let s (> 1) and t be defined as in Theorem 4.3 and suppose that $0 < \delta' < \lceil R_2'/q \rceil + s$, where q is a positive integer. Then if $q \mid c$, we must have that $\mathfrak{a}(\delta' + \rho s')$ is an*

element of $\{\overline{\mathfrak{a}}_4, \overline{\mathfrak{a}}_3, \overline{\mathfrak{a}}_2, \mathfrak{a}_1, \mathfrak{a}_2, \ldots, \mathfrak{a}_{t+2}\}$, *where* $\rho = \lfloor \delta/s' \rfloor + 1$ *(*> -1*),* $s' = s - 1$ *and* $\delta = \lceil R_2'/q \rceil - \delta'$.

Proof. Since $\rho = \lfloor \delta/s' \rfloor + 1$, we have $\delta < \rho s' < \delta + s$. Also, $\rho > \delta/s' = \lceil R_2'/q \rceil/s' - \delta'/s' \geq \lceil R_2'/q \rceil/s' - (\lceil R_2'/q \rceil + s')/s' = -s'/s' = -1$. Now $\delta + \delta' < \rho s' + \delta' < \delta + \delta' + s$ implies that $\lceil R_2'/q \rceil < \rho s' + \delta' < \lceil R_2'/q \rceil + s$.

By Lemma 4.6, we have $\lceil R_2'/q \rceil = \lceil cR_2/q \rceil + \gamma$, where $\gamma \in \{-1, 0, 1\}$; hence, $\lceil cR_2/q \rceil \leq \rho s' + \delta' \leq \lceil cR_2/q \rceil + s$. If $q \mid c$, then $\lceil c'R_2 \rceil \leq \rho s' + \delta' \leq \lceil c'R_2 \rceil + s$, where $c' = c/q$ is an integer.

By Corollary 4.5, we have $\mathfrak{a}(\lceil c'R_2 \rceil) \in \{\overline{\mathfrak{a}}_4, \overline{\mathfrak{a}}_3, \overline{\mathfrak{a}}_2, \mathfrak{a}_1, \mathfrak{a}_2, \mathfrak{a}_3, \mathfrak{a}_4\}$. Furthermore, if we put $\mathfrak{a}_u = \mathfrak{a}(s)$, then $u < t$ by Proposition 3.5. Hence, when $\mathfrak{a}_w = \mathfrak{a}(\lceil c'R_2 \rceil + s)$ we have $u - 2 \leq w \leq u + 3 \leq t + 2$ by Theorem 4.1. This shows that $\mathfrak{a}(\delta' + \rho s') \in \{\overline{\mathfrak{a}}_4, \overline{\mathfrak{a}}_3, \overline{\mathfrak{a}}_2, \mathfrak{a}_1, \mathfrak{a}_2, \ldots, \mathfrak{a}_{t+2}\}$. □

In order to use the theorem we must have $\delta' < \lceil R_2'/q_i \rceil$ after every step of the algorithm. We show that this is the case if we put $\delta' := \lceil R_2'/q_{i-1} \rceil$ after applying AX (assuming that R_2' is replaced by R_2'/q_{i-1} whenever we find a factor q_{i-1} of c) and put $\delta' := \delta' + \rho s'$ after using our approximation method.

After using AX we get $\mathfrak{a}(\lceil R_2'/q_{i-1} \rceil)$ for some $q_{i-1} \leq M$ and get $\delta' = \lceil R_2'/q_{i-1} \rceil < \lceil R_2'/q_i \rceil < \lceil R_2'/q_i \rceil + s$. Now, when we continue with the next prime $q_i < q_{i-1}$ using our approximation method, we have $\delta' < \lceil R_2'/q_{i-1} \rceil + s < \lceil R_2'/q_i \rceil + s$, so that the conditions of the theorem are still satisfied. In addition, the proof of the theorem shows that for our new distance $\delta' + \rho s'$ we have $\delta' + \rho s' < \lceil R_2'/q_i \rceil + s$ as well. Therefore we know that we can use the theorem all the way throughout the algorithm.

5.3. **Parallelization.** Because the computations required for the verification can become very time-consuming, we need to have the ability to run the algorithm in parallel. Being able to do this allows us to share the workload among multiple machines and/or processors and thereby to handle verification for number fields $\mathbb{Q}(\sqrt{D})$ where the value for D is larger than the values that we can handle with a non-parallelized version of the algorithm.

Dividing the workload for the parts of the algorithm can be done in a relatively simple way, as all parts basically work with intervals that can simply be divided into smaller, equally sized intervals. For example, while computing the baby step list we work over an interval of distances with lower bound 1 and upper bound $s + \lambda$, where we start at ideal $\mathfrak{a}(1)$ and determine when we are done by checking whether we encounter ideal $\mathfrak{a}(s + \lambda)$. To divide this interval into y smaller intervals, we introduce lower and upper bounds $(i - 1)(s + \lambda)/y$ and $i(s + \lambda)/y$ for each new interval $i = 1, \ldots, y$ and compute the corresponding ideals with distances close to these bounds using AX. If we work with gaps in the baby step lists (see Section 5.1), we also need to ensure that the number of omitted ideals is not larger than l ideals by storing the ideal corresponding with the lower bound in the list $\mathcal{L}'$. Because this may introduce additional ideals to store, we need to adjust l accordingly before initiating the computation. Similar techniques can be used for the other parts of the algorithm, for which identifying the intervals is more straightforward.

In order to run the algorithm in parallel, we employ two different techniques. The first technique is called *message passing*, which can be used to make the program run in parallel on multiple machines and/or processors using a separate process for every machine and/or processor. The processes then communicate with each other

by sending messages. The easiest way to implement a parallelized algorithm using this scheme is to make one process control the division of the tasks over all other processes and let this same process collect the results of the processes that are done with their part of the computation.

The second technique, which is mainly useful for machines with more than one processor, is called *threading*. It enables us to make a program run in parallel on multiple processors of one machine using *threads*. Threads are processes that cost the operating system very little overhead to create and execute and have the additional advantage that all threads that we create during the execution of the algorithm share the memory that is available for the algorithm, whereas the processes that are created on a machine using message passing have to divide the available memory among themselves. However, having threads share memory has the disadvantage that threads can also write to parts of memory that are simultaneously being used by other threads and can thereby invalidate the computations of another thread. Therefore, extra precautions have to be taken in order to prevent this from happening. The implementation of the verification algorithm that we created gives the user the option to choose between executing using only message passing, using only threading (when run on a machine with more than one processor) and using both message passing and threading, where each machine runs a separate process that is subsequently split into a number of threads.

The part of the verification algorithm that computes the multiplier is easy to parallelize, as each processor simply works on a different interval of primes. We employed a heuristic method to roughly balance the time spent by each processor by creating a large number of sub-intervals and supplying processors with a new interval only after they finished checking the previous interval obtained. Because an interval containing small primes takes longer to process than one containing the same number of large primes, the intervals with small primes are distributed first.

The baby-step giant-step part of the verification algorithm is unfortunately difficult to parallelize optimally unless all processors have access to the same memory. In that case, a straightforward parallelization in which each processor computes a subset of the baby steps and a subset of the giant steps would work. This approach will most likely fail in the message passing model because the communication overhead would be too high. For example, in order to determine whether a giant step ideal is in the list of baby steps, a machine would have to query every other machine to determine whether the ideal is in that machine's piece of the baby step list. As a result, we have opted for a sub-optimal solution in which each machine computes an identical copy of the baby step list. The disadvantage is that the number of baby steps is confined to that which a single machine can store, but the advantage is that there is no interprocessor communication other than the coordinating processor sending initialization information to the other processors at the beginning of the computation. If the network consists of a number of machines that each have multiple processors, then the identical baby step lists on each machine can be computed in parallel using threading within the machines.

In [5] we show that using n machines with r processors each and both techniques for parallelization we can expect a speed-up of $n^{2/3}r$. As mentioned above, the reason that we cannot obtain an optimal speed-up nr here is due to the fact that every machine has to compute its own baby step list in our approach. Most interesting is

that this estimate shows that we can expect a speed-up of r using a shared memory multiprocessor machine with r processors, but we have not yet experimented with this approach.

5.4. **Determining optimal values for s, Q and K.** In order to make an implementation of the verification algorithm efficient, it is necessary to properly balance the workload of the baby step/giant step part of the algorithm with that of the part for determining the multiplier. For this purpose, we describe a fine-grained optimization method in [5]. Here, we present an overview of this method.

Balancing the parts of the algorithm boils down to choosing values for the variables s, Q, and K that are described in previous sections. In order to get appropriate values for these variables, we introduce some additional variables and construct formulas that approximate the run times of the parts of the algorithm. For instance, because we express the expected run times of the algorithms relative to the (average) time required for computing a baby step, we need to introduce a variable that expresses the ratio between the average time needed for computing a baby step and the time needed for computing a giant step. This ratio differs for every real quadratic number field for which we verify the regulator. Therefore, we have to either perform heuristic measuring before every computation or perform a range of heuristic measurements for a variety of number fields and use these to extrapolate approximations for the other fields. In our implementation, we use the former method.

After constructing the formulas that approximate the run times, which can be found in detail in [5], we first optimize the running time for the baby step/giant step part of the algorithm. We need to do this because the formula that approximates the running time for computing the multiplier is quite complex and we cannot simultaneously determine values for all three variables s, Q and K. Since we only require $2sQ \geq K$, we can put $Q := K/(2s)$ and optimize s in terms of K. After substituting this optimal value for s, we obtain formulas for the running times of all parts of the algorithm that consist of constants and the variable K (recall that we can pick $M = R_2'/K$ while determining the value of the multiplier). We then add these formulas to get a formula for the total running time of the algorithm and determine the first and second derivative of this formula. Finally, using Newton iteration, we can then (numerically) determine an optimal value for K by determining the zero of the derivative, which corresponds to the minimum of the formula.

6. Precision

We now give an overview of the results that specify which restrictions we need to impose on the value of the precision p in order to have $f < 2^{p-4}$ for all (f, p) representations that are computed during the verification process. One of these restrictions depends on the values that are chosen for the variables s and Q, and we place some relative bounds on the values of s, Q and K in order to be able to refine the results. The precisions that are recommended for the various phases of the verification process are the following:

- For the baby step/giant step algorithm, if $s > 16$ and $Q > \max\{16, \log s\}$, then $f < 2^{p-4}$ for all (f, p) representations computed during the algorithm if p is chosen such that $2^p \geq 221Q^2s$.

- While determining the multiplier we first assume that $R_2' > 10^6$, $R_2'^{1/2} < K < R_2'^{5/6}$ and $\max\{16, K^{2/5}\} < s < K^{3/5}$, which does not restrict the choice for the values of the variables too much when R_2' is large. It turns out that, under these conditions, $f < 2^{p-4}$ for all (f, p) representations that are computed during the determination algorithm if $2^p > 19R_2' \log R_2'$.
- If the initial approximation R_2' needs to be refined, as mentioned at the beginning of Section 4, we assume that $R_2' > 10^6$ and require that $2^p > 21R_2' \log R_2'$ in order to assure that the refined value has a relative error smaller than 2^{p-4}.

The proofs of these statements can be found in detail in [5], where they are deduced using similar techniques to those employed in [11]. Since the proofs are repetitive and not very distinct from those in [11], they are not provided here.

7. Results

We implemented and optimized the verification algorithm described in the previous sections using NTL version 5.3.1 [21] and made use of the subexponential algorithm implemented in LiDIA version 2.0.1 [6]. For the results in this paper we have further improved the fine-grained optimization of the parameters s, Q and K that is described in [5] by incorporating into our optimization formulas the times required for table-lookups and for storing to memory. After incorporating a similar optimization into an implementation of the $D^{1/5+\epsilon}$ algorithm, we have run both algorithms on a machine with two Intel P4 Xeon 2.4 GHz processors and 2 GB of RAM, where we have again only used 1 GB out of the 2 GB of RAM during our computations so that the new results can be compared with our previous results in [5]. The resulting run times are listed in Table 1, where we have incorporated the time needed for the subexponential algorithm in the time needed for the $D^{1/6+\epsilon}$ algorithm, as the latter depends on the former for its input.

The data in Table 1 show that even though the $D^{1/5+\epsilon}$ algorithm is initially faster, the $D^{1/6}$ algorithm becomes significantly faster as soon as $D \approx 10^{20}$ or larger. In fact, the only reason why the $D^{1/5+\epsilon}$ has this initial advantage is because of the standard overhead of the subexponential algorithm, which for $D \approx 10^{15}$ can be seen to be the most costly part of the $D^{1/6+\epsilon}$ algorithm. Furthermore, we can see from the table that, even though the $D^{1/6+\epsilon}$ algorithm is significantly faster than the $D^{1/5+\epsilon}$ algorithm for large D, it is still very slow compared to the subexponential algorithm.

We also implemented a parallel version of the $D^{1/6+\epsilon}$ algorithm. In [5], there is a distinction between parallelization using message passing and parallelization using both message passing and threading. Our latest experiments have shown that using threading does not speed up the implementation under current conditions, so we have discontinued our efforts in this area for now. Instead, because we know that our current run-time estimation overestimates the time needed for determining the multiplier, we performed some additional testing with formulas where the expected balance between the two parts of the algorithm is modified.

We ran a parallel algorithm on 180 processors in a cluster of machines with two Intel P4 Xeon 2.4 GHz processors and 2 GB of RAM, where we have only used 850 MB of RAM per processor. The times needed for verifying the regulator for discriminants of different sizes can be found in Table 2, where the last column lists the best run-times that we were able to achieve after repeatedly modifying the

TABLE 1. Comparison of the running times for the subexponential, $D^{1/6+\epsilon}$ and $D^{1/5+\epsilon}$ algorithms for computing the regulator of $\mathbb{Q}(\sqrt{D})$.

$D \approx 10^{\cdots}$	Subexponential	$D^{1/6+\epsilon}$	$D^{1/5+\epsilon}$
15	0.29 sec	0.42 sec	0.25 sec
20	0.45 sec	0.93 sec	3.65 sec
25	0.68 sec	3.20 sec	2 min, 20 sec
30	1.44 sec	14.60 sec	44 min, 26 sec
35	2.57 sec	1 min, 27 sec	2 days, 13 hrs
40	6.06 sec	6 min, 12 sec	N/A
45	26.27 sec	1 hour, 10 min	N/A
50	1 min, 27 sec	1 day, 9 hours	N/A

TABLE 2. Comparison of the running times for the regular and modified parallelized versions of the $D^{1/6+\epsilon}$ verification algorithm.

$D \approx 10^{\cdots}$	Subexp. alg.	$D^{1/6+\epsilon}$ regular	$D^{1/6+\epsilon}$ modified
35	2.57 sec	1 min, 7 sec	2.35 sec
40	6.06 sec	1 min, 23 sec	11.02 sec
45	26.27 sec	3 min, 3 sec	1 min, 20 sec
50	1 min, 27 sec	25 min, 39 sec	12 min, 8 sec
55	5 min, 39 sec	5 hours, 41 min	2 hours, 39 min
60	21 min, 44 sec	7 days, 4 hours	4 days, 9 hours

TABLE 3. Comparison of the running times for the modified parallelized version of the $D^{1/6+\epsilon}$ verification algorithm for large D.

$D \approx 10^{\cdots}$	$R \approx \ldots \times 10^{30}$	Subexp. alg.	$D^{1/6+\epsilon}$
62	3.4	1 hour, 14 min	6 days, 3 hours
63	5.2	1 hour, 37 min	8 days, 2 hours
64	8.1	2 hours, 17 min	10 days, 13 hours
65	195.6	2 hours, 5 min	102 days, 7 hours

balance for each discriminant. The actual values of the discriminants and regulators can be found in [5] and are not repeated here.

From Table 2, we can see that our unmodified optimization does not work well for small examples. Also, adjusting the previously mentioned balance can make the execution approximately twice as fast, which shows that there is also room for improvement while optimizing for the verification of larger discriminants. More details about these results can be found in [5].

Finally, we used the modified version to unconditionally compute the regulator for a number of very large values of D, using 240 processors of the cluster. The timings of the corresponding verifications can be found in Table 3, together with a

rough approximation of the corresponding regulators.[2] In particular, we were able to unconditionally compute the regulator for a 65-digit value of D, which is far beyond the capabilities of previous unconditional algorithms.

References

1. C.S. Abel, *Ein Algorithmus zur Berechnung der Klassenzahl and des Regulators reellquadratischer Ordnungen*, Ph.D. thesis, Universität des Saarlandes, Saarbrücken, Germany, 1994.
2. 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. MR1104698 (92g:11125)
3. H. Cohen, F. Diaz y Diaz, and M. Olivier, *Calculs de nombres de classes et de régulateurs de corps quadratiques en temps sous-exponentiel*, Séminaire de Théorie des Nombres (Paris), 1993, pp. 35–46. MR1263522 (94m:11151)
4. ———, *Subexponential algorithms for class and unit group computations*, Journal Symbolic Computation **24** (1997), 434–441. MR1484490 (98m:11138)
5. R. de Haan, *A fast, rigorous technique for verifying the regulator of a real quadratic field*, Master's thesis, Universiteit van Amsterdam, Amsterdam, May 2004.
6. The 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.
7. J.L. Hafner and K.S. McCurley, *A rigorous subexponential algorithm for computation of class groups*, Journal American Mathematical Society **2** (1989), 837–850. MR1002631 (91f:11090)
8. M.J. Jacobson, Jr., R.F. Lukes, and H.C. Williams, *An investigation of bounds for the regulator of quadratic fields*, Experimental Mathematics **4** (1995), no. 3, 211–225. MR1387478 (97d:11173)
9. M.J. Jacobson, Jr., Á. Pintér, and P.G. Walsh, *A computational approach for solving* $y^2 = 1^k + 2^k + \cdots + x^k$, Mathematics of Computation **72** (2003), no. 244, 2099–2110. MR1986826 (2004c:11241)
10. 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, Walter de Gruyter, Warsaw, 2001, pp. 89–112. MR1881630 (2003f:94062)
11. ———, *An improved real quadratic field based-key exchange procedure*, Journal of Cryptology **19** (2006), 211–239. MR2213407 (2006k:94089)
12. A.K. Lenstra and H.W. Lenstra, Jr., *Algorithms in number theory*, Tech. Report 87-008, University of Chicago, 1987.
13. ———, *Algorithms in number theory*, Handbook of theoretical computer science **A** (1990), 673–715. MR1127178
14. H.W. Lenstra, Jr., *On the calculation of regulators and class numbers of quadratic fields*, London Mathematical Society Lecture Notes Series **56** (1982), 123–150. MR0697260 (86g:11080)
15. ———, *Solving the pell equation*, Notices of the American Mathematical Society **49** (2002), no. 2, 182–192. MR1875156 (2002i:11028)
16. P. Lévy, *Sur le développement en fraction continue d'un nombre choisi au hasard*, Compositio Math. **3** (1936), 286–303.
17. K.S. McCurley, *Cryptographic key distribution and computation in class groups*, Proceedings NATO ASI on Number Theory and Applications (Dordrecht), ASI series C, vol. 265, Kluwer, 1989, pp. 459–479. MR1123090 (92e:11149)
18. M. Pohst and H. Zassenhaus, *Über die Berechnung von Klassenzahlen und Klassengruppen algebraischer Zahlkörper*, J. reine angew. Math. **361** (1985), 50–72. MR0807252 (87g:11147)

[2]The run time for the 65-digit value of D may seem excessive in light of the complexity discussion from Section 4.3, which predicts an increase in run time of approximately a factor three over the 64-digit value. However, this is due to some practical issues. For the 65-digit example, the time required for l table-lookups exceeds the time used by the ADDX-operation, which causes the time for all but the baby step part to increase quadratically. Furthermore, a significant amount of computation time was added due to synchronization after each part of the verification algorithm and hardware problems in combination with insufficiently fine-grained checkpointing. Under ideal circumstances, we expect the same computation to take approximately 83 days.

19. D. Shanks, *The infrastructure of a real quadratic field and its applications*, Proceedings 1972 Number Theory Conference (Boulder), 1972. MR0389842 (52:10672)
20. ———, *On Gauss and composition*, Number Theory and Applications (NATO-Advanced Study Institute, Banff, 1988) (R.A. Mollin, ed.), Kluwer Academic Publishers Dordrecht, 1989, pp. 163–204. MR1123074 (92e:11150)
21. V. Shoup, *NTL: A library for doing number theory*, Software, 2001, Available from http://www.shoup.net/ntl.
22. A.J van der Poorten, H.J.J. te Riele, and H.C. Williams, *Computer verification of the Ankeny-Artin-Chowla conjecture for all primes less than 100 000 000 000*, Mathematics of Computation **70** (2000), no. 235, 1311–1328. MR1709160 (2001j:11125)
23. U. Vollmer, *An accelerated Buchmann algorithm for regulator computation in real quadratic fields*, Algebraic Number Theory (Sydney, Australia), July 2002, pp. 148–162. MR2041080 (2005d:11184)
24. H.C. Williams, *A numerical investigation into the length of the period of the continued fraction expansion of* $\sqrt{D}$, Mathematics of Computation **36** (1981), 593–601. MR0606518 (82f:10011)
25. ———, *Solving the Pell equation*, Proceedings Millennial Conference on Number Theory (A.K. Peters, ed.), Natick MA, 2002, pp. 397–435. MR1956288 (2003m:11051)
26. H.C. Williams and M.C. Wunderlich, *On the parallel generation of the residues for the continued fraction algorithm*, Mathematics of Computation **48** (1987), no. 177, 405–423. MR0866124 (88i:11099)

Centre for Information Security and Cryptography, University of Calgary, 2500 University Drive NW, Calgary, Alberta, Canada T2N 1N4

Current address: Centrum voor Wiskunde en Informatica, Kruislaan 413, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands

E-mail address: R.de.Haan@cwi.nl

Centre for Information Security and Cryptography, University of Calgary, 2500 University Drive NW, Calgary, Alberta, Canada T2N 1N4

E-mail address: jacobs@cpsc.ucalgary.ca

Centre for Information Security and Cryptography, University of Calgary, 2500 University Drive NW, Calgary, Alberta, Canada T2N 1N4

E-mail address: williams@math.ucalgary.ca