An Implementation of the Number Field Sieve

Marije Elkenbracht-Huizing

CONTENTS

The Number Field Sieve (NFS) is the asymptotically fastest known factoring algorithm for large integers. This article describes an implementation of the NFS, including the choice of two quadratic polynomials, both classical sieving and a special form of lattice sieving (line sieving), the block Lanczos method and a new square root algorithm. Finally some data on factorizations obtained with this implementation are listed, including the record factorization of $12^{151} - 1$.

1. INTRODUCTION

The Number Field Sieve (NFS), introduced in 1988 [Pollard 1993a], is the asymptotically fastest known algorithm for factoring integers. Two forms of the NFS have been considered: the Special NFS, or SNFS, tailored especially to integers of the form $n = c_1 r^t + c_2 s^u$, and the General NFS, or GNFS, applicable to arbitrary numbers. The NFS factors integers n in heuristic time

$$\exp\bigl((c + o(1))(\log n)^{1/3}(\log\log n)^{2/3}\bigr)$$

as $n \to \infty$, where $c = \left(\frac{32}{9}\right)^{1/3} \approx 1.5$ for the SNFS and $c = \left(\frac{64}{9}\right)^{1/3} \approx 1.9$ for the GNFS [Buhler et al. 1993]. These expressions should be compared with the time

$$\exp\bigl((1 + o(1))(\log n)^{1/2}(\log\log n)^{1/2}\bigr)$$

taken by the Multiple Polynomial Quadratic Sieve, or MPQS [Pomerance 1985], still the best general-purpose factoring algorithm for integers with less than approximately 105 digits.

We describe here several experiments carried out with an implementation of the NFS written by J. Buhler, R. M. Elkenbracht-Huizing, P. L. Montgomery, R. Robson and R. Ruby. It has been used, among others, for the record SNFS factorization

AMS Subject Classification (1991): 11Y05, 11Y40

Keywords: number field sieve, factorization.

This research is funded by the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (Netherlands Organization for Scientific Research, NWO) through the Stichting Mathematisch Centrum (SMC), under grant number 611-307-022.

of $(12^{151}-1)/11$, a number of 162 decimal digits, and a GNFS factorization of a 107-digit cofactor of $6^{223}+1$. We start with a description of the NFS and an outline of the implementation, then discuss in more detail several aspects of the implementation, and finally state the results of the factorization experiments. Detailed descriptions of the NFS can be found in [Lenstra et al. 1993b; Buhler et al. 1993].

2. DESCRIPTION OF THE NFS

Let n be the odd number to be factored. It is easy to check whether n is a prime number or a prime power [Lenstra et al. 1993c, § 2.5], and we assume that it is neither. Like MPQS, the NFS tries to find a solution of the equation $v^2 \equiv w^2 \bmod n$. For at least half of the pairs ($v \bmod n$, $w \bmod n$) with $v^2 \equiv w^2 \bmod n$ and v and w relatively prime to n, the greatest common divisor of n and $v-w$ gives a nontrivial factor of n.

To construct v and w we first choose two polynomials

$$\begin{aligned} f_1(x) &= c_{1,d_1}x^{d_1} + c_{1,d_1-1}x^{d_1-1} + \cdots + c_{1,0} \\ f_2(x) &= c_{2,d_2}x^{d_2} + c_{2,d_2-1}x^{d_2-1} + \cdots + c_{2,0} \end{aligned}$$

over $\mathbb{Z}$, with $f_1 \neq \pm f_2$, both irreducible over $\mathbb{Z}$ and having *content* $\operatorname{cont} f_i := \gcd(c_{i,d_i}, \ldots, c_{i,0})$ equal to 1; we also choose an integer m that is a common root modulo n of f_1 and f_2. In our implementation this is the only step in which the SNFS and the GNFS differ: in the SNFS we use the special form of n to pick these polynomials by hand. One polynomial will have very small coefficients compared to the coefficients of the polynomials we will use with the GNFS, where we search for a pair of polynomials with help of the computer. This makes SNFS faster than GNFS [Buhler et al. 1993, § 1]. See Section 5 for a detailed description of the selection of the polynomials.

Let α_i, for $i = 1, 2$, be a root of $f_i(x)$ in $\mathbb{C}$. Let $\mathbb{Q}_n$ denote the ring of rational numbers with denominator coprime to n. We want to find a nonempty set $\mathcal{S}$ of pairs (a, b) of coprime integers such that both $\prod_{\mathcal{S}}(a-b\alpha_1)$ and $\prod_{\mathcal{S}}(a-b\alpha_2)$ are squares — β^2 and γ^2, say — in $\mathbb{Q}_n[\alpha_1]$ and $\mathbb{Q}_n[\alpha_2]$, respectively. Applying to β^2 and γ^2 the two natural ring homomorphisms $\varphi_i : \mathbb{Q}_n[\alpha_i] \to \mathbb{Z}/n\mathbb{Z}$ determined by $\varphi_i(\alpha_i) = m \bmod n$ gives $\varphi_1(\beta^2) \equiv \varphi_2(\gamma^2) \bmod n$. This yields $\varphi_1(\beta)^2 \equiv \varphi_2(\gamma)^2 \bmod n$. When $\varphi_1(\beta)$ and $\varphi_2(\gamma)$ are relatively prime to n, calculating $\gcd(n, \varphi_1(\beta) - \varphi_2(\gamma))$ will yield a nontrivial factor of n in at least half the cases.

For $\prod_{\mathcal{S}}(a-b\alpha_i)$ to be a square in $\mathbb{Q}_n[\alpha_i]$, its norm $N(\prod_{\mathcal{S}}(a-b\alpha_i))$ must be a square in $\mathbb{Q}$. Denote by $F_i(x,y) = y^{d_i} f_i(x/y) \in \mathbb{Z}[x,y]$ the homogeneous form of $f_i(x)$. From $N(a-b\alpha_i) = F_i(a,b)/c_{i,d_i}$ we can deduce that if the cardinality of the set $\mathcal{S}$ is even and if $\prod_{\mathcal{S}} F_i(a,b)$ is a square in $\mathbb{Z}$, then $N(\prod_{\mathcal{S}}(a-b\alpha_i))$ is a square in $\mathbb{Q}$.

The algorithm searches for a pair (a, b) of coprime integers such that both integers $F_i(a,b)$ factor completely over the prime numbers below some user-determined bounds B_i. We call such integers $F_i(a,b)$ *smooth* and such (a,b)-pairs *relations.* For a relation (a_j, b_j) we can write

$$\begin{aligned} F_1(a_j,b_j) &= \prod_{p\in\mathcal{K}_1} p^{e_1(j,p)}, \\ F_2(a_j,b_j) &= \prod_{p\in\mathcal{K}_2} p^{e_2(j,p)}, \end{aligned} \tag{2.1}$$

where $e_i(j,p) \in \mathbb{N}$, for $i = 1, 2$, and where $\mathcal{K}_1$ and $\mathcal{K}_2$ contain -1 and the prime numbers below B_1 and B_2, respectively.

In order for $\prod_{\mathcal{S}} F_i(a,b)$ to be a square in $\mathbb{Z}$, every exponent $\sum_{\mathcal{S}} e_i(j,p)$ in

$$\prod_{(a_j,b_j)\in\mathcal{S}} F_i(a_j,b_j) = \prod_{p\in\mathcal{K}_i} p^{\sum_{\mathcal{S}} e_i(j,p)}$$

should be even. Let $\boldsymbol{v}(a_j,b_j)$ be a vector of length $1 + |\mathcal{K}_1| + |\mathcal{K}_2|$, constructed as follows: its first entry is 1 and the rest of $\boldsymbol{v}(a_j,b_j)$ is filled with all exponents $e_1(j,p)$ and $e_2(j,p)$ modulo 2, in an order which is fixed for all (a_j,b_j). If $\mathcal{S}$ is a subset of the relations such that $\sum_{(a,b)\in\mathcal{S}} \boldsymbol{v}(a,b) \equiv \boldsymbol{0} \bmod 2$, then the cardinality of $\mathcal{S}$ is even and $\sum_{\mathcal{S}} e_i(j,p) \equiv 0 \bmod 2$ for $i = 1, 2$ and all $p \in \mathcal{K}_i$; hence both $N(\prod_{\mathcal{S}}(a-b\alpha_i))$ are squares in $\mathbb{Q}$.

Unfortunately $N(\prod_{\mathcal{S}}(a-b\alpha_i))$ being a square in $\mathbb{Q}$ is not sufficient for $\prod_{\mathcal{S}}(a-b\alpha_i)$ to be a square in $\mathbb{Q}_n[\alpha_i]$. By looking at what kind of p divides $F_i(a,b)$, we will almost overcome this problem. For each prime number p we define the set

$$\mathcal{R}_i(p)=\{(r_1\colon r_2)\in \mathbf{P}^1(\mathbb{F}_p)\mid F_i(r_1,r_2)\equiv 0 \bmod p\}, \tag{2.2}$$

where $\mathbf{P}^1(\mathbb{F}_p)$ denotes the projective line over $\mathbb{F}_p$. For a and b coprime, the integer $F_i(a,b)$ is divisible by a prime number p if and only if $(a \bmod p : b \bmod p)\in\mathcal{R}_i(p)$. Therefore the set $\mathcal{R}_i(p)$ is partitioning all (a,b)-pairs for which p divides $F_i(a,b)$ according to $(a \bmod p : b \bmod p)$.

Next, for $C\in\mathbb{N}$, let $\mathcal{F}_i(C)$ be the set of pairs $(p,\,(r_1\colon r_2))$, where p is a prime less than C and $(r_1\colon r_2)\in\mathcal{R}_i(p)$. Heuristically, $|\mathcal{F}_i(C)|$ is approximately the number of primes below C [Lang 1970, Chapter VIII, § 4]. $\mathcal{F}_1(B_1)$ and $\mathcal{F}_2(B_2)$ are called the *factor bases*. We now can write (2.1) as

$$F_i(a_j,b_j)=\pm\prod_{(p,(r_1:r_2))\in\mathcal{F}_i(B_i)} p^{e_i(j,p,r_1,r_2)}, \tag{2.3}$$

for $i=1,2$, where $e_i(j,p,r_1,r_2)=e_i(j,p)$ if

$$(a_j \bmod p : b_j \bmod p)=(r_1\colon r_2)$$

and 0 otherwise.

In order for $\prod_{\mathcal{S}}(a-b\alpha_i)$ to be a square in $\mathbb{Q}_n[\alpha_i]$, every exponent $\sum_{\mathcal{S}} e_i(j,p,r_1,r_2)$ in

$$\prod_{(a_j,b_j)\in\mathcal{S}} F_i(a_j,b_j)=\pm\prod_{(p,(r_1:r_2))\in\mathcal{F}_i(B_i)} p^{\sum_{\mathcal{S}} e_i(j,p,r_1,r_2)}$$

should be even. Let $\boldsymbol{v}(a_j,b_j)$ be a vector of length $1+|\mathcal{F}_1(B_1)|+|\mathcal{F}_2(B_2)|$ containing 1 and the values of $e_1(j,p,r_1,r_2) \bmod 2$ and $e_2(j,p,r_1,r_2) \bmod 2$, in an order that is fixed for all relations (a_j,b_j). A nonempty subset $\mathcal{S}$ of relations such that

$$\sum_{\mathcal{S}} \boldsymbol{v}(a,b)\equiv \mathbf{0} \bmod 2$$

is almost sufficient to ensure that $\prod_{\mathcal{S}}(a-b\alpha_i)$ be a square in $\mathbb{Q}_n[\alpha_i]$ for $i=1,2$ [Buhler et al. 1993, § 12.7]. That it is not totally sufficient is only partly caused by the fact that we only forced the product $\prod_{\mathcal{S}}|F_i(a,b)|$ to be a square in $\mathbb{Z}$. We can see that it is not totally sufficient from the following example: In the field $\mathbb{Q}(\sqrt{3})$ generated by a root of the polynomial $f(x)=x^2-3$, the element $2+\sqrt{3}$ has norm $F(2,-1)=1$. So all exponents $e_1(j,p,r_1,r_2)$ and $e_2(j,p,r_1,r_2)$ will be zero. Furthermore $\boldsymbol{v}(2,-1)+\boldsymbol{v}(1,0)\equiv\mathbf{0} \bmod 2$. But the square root of $2+\sqrt{3}$ is $(\sqrt{6}+\sqrt{2})/2$, which is not an element of $\mathbb{Q}(\sqrt{3})$.

The small gap between being almost a square and being practically certainly a square is overcome by using *quadratic characters*, following an idea of Adleman [1991]. For $\mathcal{S}$ a set consisting of pairs (a,b) of coprime integers, let $\prod_{\mathcal{S}}(a-b\alpha_i)$ be a square in $\mathbb{Q}_n[\alpha_i]$, and let q be an odd prime number not dividing $c_{1,d_1}c_{2,d_2}$. If $(s_1\colon s_2)\in\mathcal{R}_i(q)$ is such that $f_i'(s_1s_2^{-1} \bmod q)\not\equiv 0 \bmod q$ and $(a \bmod q : b \bmod q)\neq(s_1\colon s_2)$ for all $(a,b)\in\mathcal{S}$, then

$$\prod_{(a,b)\in\mathcal{S}}\left(\frac{a-b\ (s_1s_2^{-1} \bmod q)}{q}\right)=1 \tag{2.4}$$

where $\left(\frac{v}{w}\right)$ denotes the Legendre symbol [Buhler et al. 1993, § 8, § 12.7]. We use this by taking for each polynomial several primes q larger than B_i and not dividing c_{i,d_i}, together with an element $(s_1\colon s_2)\in\mathcal{R}_i(q)$ such that $f_i'(s_1s_2^{-1} \bmod q)\not\equiv 0 \bmod q$. Since $q>B_i$ we have $(a \bmod q : b \bmod q)\neq(s_1\colon s_2)$ for all relations (a,b). Append to the vector $\boldsymbol{v}(a,b)$ for all pairs $(q,(s_1\colon s_2))$ a 0 if

$$\left(\frac{a-b\ (s_1s_2^{-1} \bmod q)}{q}\right)=1$$

and a 1 otherwise. Now a nonempty subset $\mathcal{S}$ of all relations such that $\sum_{\mathcal{S}}\boldsymbol{v}(a,b)\equiv\mathbf{0} \bmod 2$ guarantees that (2.4) holds for all chosen primes q together with their elements $(s_1\colon s_2)\in\mathcal{R}_i(q)$. Taking enough quadratic characters — we took 32 per polynomial — makes it practically certain that both $\prod_{\mathcal{S}}(a-b\alpha_i)$ are squares in $\mathbb{Q}_n[\alpha_i]$. The counterexample given earlier could have been caught with the use of quadratic characters: take $q=11$ and $(s_1\colon s_2)=(5,1)\in\mathcal{R}(11)$. The Legendre symbol becomes $\left(\frac{2+5}{11}\right)$, which is -1.

If Q is the total number of quadratic characters used, then a nonempty subset $\mathcal{S}$ such that $\sum_{\mathcal{S}} \boldsymbol{v}(a,b) \equiv \mathbf{0} \bmod 2$ can always be found if the number of relations exceeds

$$1 + |\mathcal{F}_1(B_1)| + |\mathcal{F}_2(B_2)| + Q.$$

3. OUTLINE OF THE IMPLEMENTATION

The implementation can be divided into five stages. In the first stage we select the polynomials $f_1(x)$ and $f_2(x)$ in $\mathbb{Z}[x]$, and the integer m such that m is a common root of $f_1(x)$ and $f_2(x)$ modulo n. We also choose the *sieving region*—that is, the collection of (a,b)-pairs for which both $F_i(a,b)$ are checked for smoothness—and, for each polynomial, a *factor base bound* B_i.

The second stage, the sieving in which the relations are found, is the most time-consuming. In this implementation a relation is a pair (a,b) from the sieving region such that both $F_i(a,b)$ factor completely over the primes below B_i, except for at most two large prime numbers, which should be between B_i and a *large prime bound* L_i. By using lattice sieving [Pollard 1993b]—a special form of which will be desribed in Section 6—one of the two integers $F_i(a,b)$ is allowed to have three primes between B_i and L_i. The product in (2.3) is taken over $\mathcal{F}(L_i)$, and the vectors $\boldsymbol{v}(a,b)$ have to be adapted accordingly.

This is followed by a filtering stage with the purpose of reducing the amount of data. Here some relations are eliminated and others are grouped into relation-sets.

In the fourth stage, we construct a matrix by taking the vectors $\boldsymbol{v}(a,b)$ and the vectors

$$\sum_{(a,b)\in\mathcal{V}} \boldsymbol{v}(a,b)$$

for all remaining relations and relation-sets $\mathcal{V}$ as columns. Finding a nonempty set $\mathcal{S}$ such that

$$\sum_{\mathcal{S}} \boldsymbol{v}(a,b) \equiv \mathbf{0} \bmod 2$$

is the same as calculating a nontrivial vector from the null space of this matrix over $\mathbb{F}_2$. For huge sparse matrices the best known methods are iterative ones, such as the block Lanczos algorithm [Montgomery 1995]. The output of this stage is a subset $\mathcal{S}$ of the relations such that both $\prod_{\mathcal{S}}(a-b\alpha_1)$ and $\prod_{\mathcal{S}}(a-b\alpha_2)$ are squares β^2 and γ^2 in $\mathbb{Q}_n[\alpha_1]$ and $\mathbb{Q}_n[\alpha_2]$, respectively.

The final stage consists of extracting the square roots β and γ. This is done by a new algorithm, developed by Montgomery [1994] and also iterative. Successive approximations are found, leaving over "smaller" remainders of which we have to extract the square root. If the remainder is small enough we use a conventional method. Finally we apply the homomorphisms φ_1 and φ_2 to the square roots β and γ, respectively, and calculate the gcd of n and $\varphi_1(\beta)-\varphi_2(\gamma)$, which will split n into two nontrivial factors in at least half of the cases.

4. FREE RELATIONS

Denote the order of the Galois group of $f_1(x)f_2(x)$ by g. For approximately $1/g$ of the primes $q < \min(L_1, L_2)$, both polynomials $F_i(x,y)$ split into d_i linear factors modulo q [Frobenius 1896, § 2, Theorem 1; Neukirch 1992, p. 566]:

$$F_i(x,y) = c_{i,d_i}\prod_{j=1}^{d_i}\bigl(r_2^{(j)}x - r_1^{(j)}y\bigr) \bmod q \qquad (4.1)$$

If such a prime q does not divide the discriminants of $f_1(x)$ or $f_2(x)$ (and therefore both polynomials $F_i(x,y)$ split into d_i *different* linear factors modulo q) and if q does not divide $c_{1,d_1}\cdot c_{2,d_2}$, we call q a *free prime*. This terminology comes from the fact that we can select such primes that are smaller than $\min(B_1,B_2)$ without extra effort when calculating the factor bases $\mathcal{F}_i(B_i)$. They are said to give rise to free relations because we now require $\bigl(\prod_{p\in\mathcal{T}}p\bigr)\bigl(\prod_{(a,b)\in\mathcal{S}}(a-b\alpha_i)\bigr)$ to be a square in $\mathbb{Q}_n[\alpha_i]$, for $i=1,2$, where $\mathcal{T}$ is a suitably chosen subset of the set of free primes. With $N(p)=p^{d_i}$, we have

$$N\Big(\Big(\prod_{p\in\mathcal{T}} p\Big)\Big(\prod_{(a,b)\in\mathcal{S}}(a-b\alpha_i)\Big)\Big) = \Big(\prod_{p\in\mathcal{T}} p^{d_i}\Big)\Big(\prod_{(a,b)\in\mathcal{S}}\frac{F_i(a,b)}{c_{i,d_i}}\Big),$$

which represents a square in $\mathbb{Q}$ if $|\mathcal{S}|$ is even and $\big(\prod_{p\in\mathcal{T}} p^{d_i}\big)\big(\prod_{(a,b)\in\mathcal{S}} F_i(a,b)\big)$ is a square in $\mathbb{Z}$. As we partitioned the primes p dividing $\prod_{(a,b)\in\mathcal{S}}F_i(a,b)$ according to the roots $(a \bmod p : b \bmod p) \in \mathcal{R}_i(p)$, we consider p^{d_i} as the product of one factor p for every root $(r_1 : r_2) \in \mathcal{R}_i(p)$. We associate with every free prime $p < \min(L_1, L_2)$ a vector $\boldsymbol{v}(p)$ of length $1 + |\mathcal{F}_1(L_1)| + |\mathcal{F}_2(L_2)| + Q$, which contains a one for every $(p,\, (r_1^{(j)} : r_2^{(j)}))$ occurring in (4.1) for both polynomials, and for every quadratic character $(q, (s_1 : s_2))$ for which $(\frac{p}{q}) = -1$. The rest is filled with zeros. We will look for $\mathcal{T}$ and $\mathcal{S}$ such that $\sum_{p\in\mathcal{T}} \boldsymbol{v}(p) + \sum_{(a,b)\in\mathcal{S}} \boldsymbol{v}(a,b) \equiv \boldsymbol{0} \bmod 2$.

5. CHOICE OF THE POLYNOMIALS

The conjectured running time for the application of the SNFS to a number of the form $n = c_1 r^t + c_2 s^u$ depends on the size of n. If only small factors of n are known, the SNFS algorithm is certainly the best one to use. If already a substantial nonalgebraic factor of n is known, the GNFS or the MPQS might be faster.

Using the SNFS for a factor n of an integer $c_1 r^t + c_2 s^u$ with $\gcd(c_1 r, c_2 s) = 1$, we pick the two possibly nonmonic polynomials by hand. Select a small positive integer d_1 — usually 4 or 5 — which will be the degree of $f_1(x)$. Write $t = d_1 t' + t''$ and $u = d_1 u' + u''$ with $t'', u'' \in \{0, 1, \dots, d_1 - 1\}$. In practice $f_1(x) := c_2 s^{u''} x^{d_1} + c_1 r^{t''}$ is irreducible over $\mathbb{Z}$, and $f_1(x)$, $f_2(x) := r^{t'} x - s^{u'}$, and $m := s^{u'} r^{-t'} \bmod n$ satisfy the requirements mentioned in Section 2. If $f_1(x)$ is not irreducible, a nontrivial factor of f_1 is likely to give rise to a nontrivial factor of n, and otherwise f_1 can be replaced by a suitable factor. (This is also applicable in the case of the GNFS.) An algorithm to test whether a polynomial is irreducible and to factor it if it is not can be found in [Lenstra et al. 1982]. If we encounter a polynomial $f_i(x)$ with $\operatorname{cont} f_i(x) \neq 1$, we can divide all coefficients of $f_i(x)$ by the content, assuming that $\operatorname{cont} f_i(x)$ and n are relatively prime. Using the SNFS we sometimes find better pairs of polynomials, together with a value for m, by trying to factor a multiple of n. Examples can be found in the last section of this article.

Using the GNFS one can find two polynomials by the *base m* method. Select a small positive integer d_1 — usually 4 or 5 — which will again be the degree of $f_1(x)$. Set $m = \lfloor n^{1/d_1} \rfloor$ and write n in base m as

$$n = c_{d_1} m^{d_1} + c_{d_1-1} m^{d_1-1} + \cdots + c_0,$$

with $0 \le c_i < m$. Now

$$f_1(x) = c_{d_1} x^{d_1} + c_{d_1-1} x^{d_1-1} + \cdots + c_0$$

and $f_2(x) = x - m$ satisfy the requirements. This method implies $c_{d_1} = 1$ [Buhler et al. 1993, § 3]. In [Buhler et al. 1993, § 12.2] one can find slightly better variants of this method, resulting in a linear and a higher-degree polynomial with leading coefficients possibly larger than one and possibly negative coefficients. For these variants the polynomial coefficients are $\mathcal{O}(n^{1/(d_1+1)})$.

The task is to find suitable polynomials f_1 and f_2, factor base bounds B_1, B_2, large prime bounds L_1, L_2, and a sieving region. For a good choice four characteristics of the polynomials should be taken into account. First, the maximal values of $|F_i(a,b)|$ should be small, making them more likely to be smooth over the primes below B_i. Secondly, when a polynomial has many real roots, more ratios a/b will be near a root and more values $F_i(a,b)$ are expected to be small. As a refinement of this characteristic we can look at the absolute value of the real roots. A polynomial having a real root near $\max|a|/\max|b|$ is a good choice. The importance of this characteristic is made clear in Figure 1. Thirdly, polynomials that have many roots modulo (preferably different) small primes are preferred over ones that do not. This enlarges the probability that $F_i(a,b)$ is small after dividing it by these small prime numbers, making it more likely to be smooth over the primes below B_i. Finally, it is better to choose polynomials for which the order of the Galois group of $f_1(x)f_2(x)$ is small, since we saw in the previous section that they provide more free relations. With these criteria in mind we select the pair of polynomials which is expected to be the best.

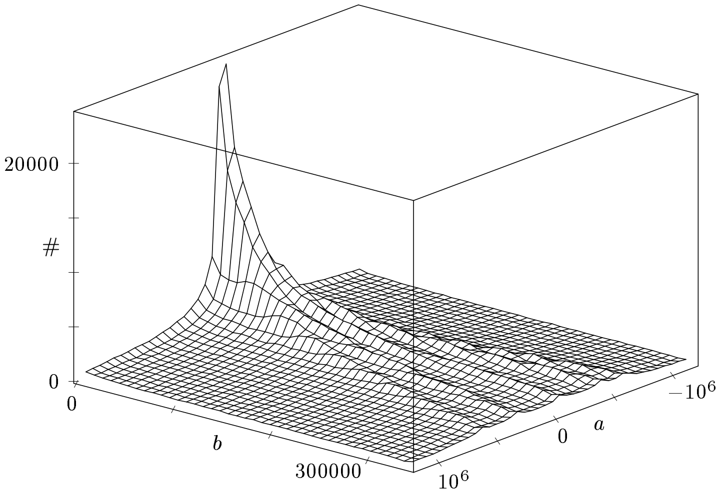

FIGURE 1. Number of relations found for 60000 a-values and 8625 b-values for the factorization of a 119-digit factor of $3^{319}-1$ (see Section 10 for details). One polynomial is $f_1(x) = x^5+x^4-4x^3-3x^2+3x+1$, with 5 real roots. The five ridges indicate a higher yield for pairs (a, b) with a/b near a root.

We experimented with a choice of two quadratic polynomials selected according to ideas of Montgomery [Buhler et al. 1994]. He observed that $f_1(x) = c_{1,2}x^2 + c_{1,1}x + c_{1,0}$ and $f_2(x) = c_{2,2}x^2 + c_{2,1}x + c_{2,0} \in \mathbb{Z}[x]$ have a common root m modulo n if and only if the vectors $\boldsymbol{a} = (c_{1,0}, c_{1,1}, c_{1,2})^T$ and $\boldsymbol{b} = (c_{2,0}, c_{2,1}, c_{2,2})^T$ are orthogonal to $(1, m, m^2)^T$ over $\mathbb{Z}/n\mathbb{Z}$ using the standard inner product. Suppose $f_1(x)$ and $f_2(x)$ are irreducible over $\mathbb{Z}$, have content 1, and do not satisfy $f_1(x) = \pm f_2(x)$. As will be explained further on, we can find in practice $\boldsymbol{a}$ and $\boldsymbol{b}$ of which the coefficients are appoximately $\mathcal{O}(n^{1/4})$, so the space orthogonal to $\boldsymbol{a}$ and $\boldsymbol{b}$ has rank 1 (both over $\mathbb{Z}$ and over $\mathbb{Z}/n\mathbb{Z}$). If $\boldsymbol{c} = \boldsymbol{a} \times \boldsymbol{b}$ (cross product), then $\boldsymbol{c}$ must be a multiple of $(1, m, m^2)^T$ over $\mathbb{Z}/n\mathbb{Z}$. The fact that $f_1(x)$ and $f_2(x)$ are not multiples of each other ensures that $\boldsymbol{c}$ is not the zero vector. If $\boldsymbol{c} = (c_0, c_1, c_2)^T$, then c_0, c_1, c_2 is a geometric progression in $\mathbb{Z}/n\mathbb{Z}$. It is not a geometric progression over $\mathbb{Z}$, since then $f_1(x)$ and $f_2(x)$ would have a common factor $x-m$ over $\mathbb{Z}$.

Montgomery's algorithm for finding $f_1(x)$ and $f_2(x)$ reverses this construction and starts with a vector $\boldsymbol{c} = (c_0, c_1, c_2)^T \in \mathbb{Z}^3$, where c_0, c_1, c_2 is a geometric progression with ratio m over $\mathbb{Z}/n\mathbb{Z}$, but not over $\mathbb{Z}$. The vector $\boldsymbol{c}$ can be constructed as follows: for p prime such that $p < \sqrt{n}$ and n a quadratic residue modulo p, choose c_1 such that $c_1^2 \equiv n \bmod p$ and $|c_1 - n^{1/2}| \le p/2$. The elements of $\boldsymbol{c} = (p, c_1, (c_1^2-n)/p)^T$ form a geometric progression with ratio c_1/p over $\mathbb{Z}/n\mathbb{Z}$, not over $\mathbb{Z}$. Furthermore $c_i = \mathcal{O}(n^{1/2})$ (i=0,1,2). Take $s \in \mathbb{Z}/p\mathbb{Z}$ such that $c_1 s \equiv 1 \bmod p$. With $c_2 = (c_1^2 - n)/p$, the vectors

$$\boldsymbol{a}' = \begin{pmatrix} c_1 \\ -p \\ 0 \end{pmatrix}, \quad \boldsymbol{b}' = \begin{pmatrix} (c_1(c_2 s \bmod p) - c_2)/p \\ -(c_2 s \bmod p) \\ 1 \end{pmatrix}$$

are both orthogonal to $\boldsymbol{c}$. From $\boldsymbol{a}' \times \boldsymbol{b}' = -\boldsymbol{c}$ and $\gcd(c_0, c_1, c_2) = 1$ we deduce that $\boldsymbol{a}'$ and $\boldsymbol{b}'$ span the sublattice of $\mathbb{Z}^3$ orthogonal to $\boldsymbol{c}$. Denote by $(\boldsymbol{a}, \boldsymbol{b})$ the inner product of $\boldsymbol{a}$ and $\boldsymbol{b}$, and remember that $\boldsymbol{a}\,(\boldsymbol{a}, \boldsymbol{b})/(\boldsymbol{a}, \boldsymbol{a})$ is the projection of $\boldsymbol{b}$ on $\boldsymbol{a}$. By reducing the basis $\{\boldsymbol{a}', \boldsymbol{b}'\}$, one can find "small" vectors $\boldsymbol{a}$ and $\boldsymbol{b}$ with $\{\boldsymbol{a}, \boldsymbol{b}\}$ a basis of the sublattice of $\mathbb{Z}^3$ orthogonal to $\boldsymbol{c}$, such that

$$\left|\frac{(\boldsymbol{a}, \boldsymbol{b})}{(\boldsymbol{a}, \boldsymbol{a})}\right| \le \tfrac{1}{2} \quad \text{and} \quad \left|\frac{(\boldsymbol{a}, \boldsymbol{b})}{(\boldsymbol{b}, \boldsymbol{b})}\right| \le \tfrac{1}{2}.$$

The angle θ between these vectors will be between 60° and 120°. Since the surface of the parallelogram spanned by $\boldsymbol{a}$ and $\boldsymbol{b}$ is both equal to $\|\boldsymbol{a} \times \boldsymbol{b}\|$ and $\|\boldsymbol{a}\| \cdot \|\boldsymbol{b}\| \sin\theta$, we have

$$\|\boldsymbol{a}\| \cdot \|\boldsymbol{b}\| = \frac{\|\boldsymbol{c}\|}{\sin\theta} \le \frac{2\|\boldsymbol{c}\|}{\sqrt{3}} = \mathcal{O}(\|\boldsymbol{c}\|) = \mathcal{O}(n^{1/2}). \tag{5.1}$$

In practice both $\|\boldsymbol{a}\|$ and $\|\boldsymbol{b}\|$ are $\mathcal{O}(n^{1/4})$. For different values of p we will get a different pair of polynomials. The program `findquad` tries to find two polynomials each having two real roots, many roots modulo small primes, and such that the integral of $\sum_{i=1,2} \log |F_i(x,y)|$ is small, where (x,y) runs through the sieving region for (a,b).

When using line sieving (a special form of lattice sieving, explained in Section 6), we like to use a large range of a-values, say $|a| < M$ and only $b = 1$. To try to induce $F_i(a,1) = c_{i,2}a^2 + c_{i,1}a + c_{i,0}$ to be smooth over the prime numbers below B_i, we would prefer $c_{i,2} = \mathcal{O}(n^{1/4}/M)$, $c_{i,1} = \mathcal{O}(n^{1/4})$, and $c_{i,0} = \mathcal{O}(n^{1/4}M)$ rather than all of them being $\mathcal{O}(n^{1/4})$. We achieve this by first choosing $c_0 = p = \mathcal{O}(\sqrt{n}/M)$, whence $c_1 = \mathcal{O}(\sqrt{n})$ and $c_2 = \mathcal{O}(\sqrt{n}M)$. The resulting coefficients of $\boldsymbol{a}' = (a_0', a_1', a_2')$ and $\boldsymbol{b}' = (b_0', b_1', b_2')$ have approximately the right ratio. To keep this ratio while reducing the basis, we reduce the vectors $\boldsymbol{a}'' = (a_0', a_1'M, a_2'M^2)$ and $\boldsymbol{b}'' = (b_0', b_1'M, b_2'M^2)$ instead. Note that $\boldsymbol{a}'' \times \boldsymbol{b}'' = M\boldsymbol{c}''$ with $\boldsymbol{c}'' = (c_0M^2, c_1M, c_2)$, which is still a geometric progression with ratio c_1/pM over $\mathbb{Z}/n\mathbb{Z}$, not over $\mathbb{Z}$. Using (5.1) with $\boldsymbol{c} = M\boldsymbol{c}''$ we find that the resulting vectors $\boldsymbol{a} = (a_0, a_1M, a_2M^2)$ and $\boldsymbol{b} = (b_0, b_1M, b_2M^2)$ will be both $\mathcal{O}(n^{1/4}M)$. Using $f_1(x) = a_2x^2 + a_1x + a_0$ and $f_2(x) = b_2x^2 + b_1x + b_0$ results in the desired orders of the coefficients of f_1 and f_2.

We also have to choose the factor base bounds B_i, the large prime bounds L_i, and the sieving region. In the experiments described in Section 10, where we factored numbers in the 98–162 digits range with the SNFS and numbers in the 87–107 digits range with the GNFS, we used factor base bounds between $5 \cdot 10^5$ and $2.9 \cdot 10^6$ and large prime bounds between $12 \cdot 10^6$ and $4 \cdot 10^7$. When using classical sieving, the sieving region was a rectangle for which we took a in a subinterval of $[-2 \cdot 10^6,\ 2 \cdot 10^6]$ and b between 1 and some upper bound, in our experiments between $16 \cdot 10^3$ and $48 \cdot 10^4$. First we chose the factor base bounds and generated the corresponding factor bases $\mathcal{F}_i(B_i)$, as described in the next section. Then we chose the large prime bounds L_i and fixed a range of a-values. For all these a-values and a few b-values, preferably equidistributed over the expected range of b-values, we checked whether the (a,b)-pair is a relation, in a way we will describe in the next section. Allowing $F_i(a,b)$ to contain two large primes between B_i and L_i, instead of demanding it to be smooth over the primes below B_i, we increase the probability that (a,b) is a relation. On the other hand, $F_i(a,b)$ is now factored over $\mathcal{F}_i(L_i)$ instead of $\mathcal{F}_i(B_i)$, which enlarges the number of relations needed too. In practice this adjustment has shown to be useful. In our experiments we needed approximately $0.8 \cdot \{\pi(L_1) + \pi(L_2) - \pi(\min(L_1, L_2))/g\}$ relations.

From the number of relations we got for these few b-values we could estimate the range of b-values needed and from the time the experiment took we could estimate the time needed for the whole sieving step. In this way we selected a good combination of pair of polynomials, factor base bounds, large prime bounds and sieving region.

6. THE SIEVING

The sieving is the part of the algorithm during which we collect the relations. Before we start the sieving we have to generate the factor bases $\mathcal{F}_i(B_i)$ defined on page 233, just before (2.3). If we identify $\mathbf{P}^1(\mathbb{F}_p)$ with $\mathbb{F}_p \cup \{\infty\}$ by identifying $(r_1 : r_2)$ with r_1/r_2, then $\mathcal{R}_i(p)$ (defined in (2.2)) consists of those $r = r_1/r_2 \in \mathbb{F}_p$ for which $f_i(r) \equiv 0 \bmod p$, together with ∞ if $c_{i,d_i} \equiv 0 \bmod p$. The program `rootfinder` finds for both polynomials $f_i(x)$ and for all primes p below B_i all roots modulo p. Repeated roots appear only once in the list. When the prime p divides the leading coefficient c_{i,d_i}, `rootfinder` includes the projective root $(1,0)$, which it represents by p. We recall that for a and b coprime, $p \mid F_i(a,b)$ if and only if $(a \bmod p : b \bmod p) \in \mathcal{R}_i(p)$. In terms of the roots of $f_i(x)$ this means that for a and b coprime, $p \mid F_i(a,b)$ if and only if ($a \equiv br \bmod p$ and $f_i(r) \equiv 0 \bmod p$) or ($p \mid c_{i,d_i}$ and $p \mid b$).

Two ways of sieving have been implemented: the "classical" sieve [Lenstra et al. 1993b, § 4; Buhler et al. 1993, § 12], and line sieving, a special form of lattice sieving [Pollard 1993b].

In the classical way of sieving we first choose the a-interval and the b-interval. We start sieving with $b = 1$ and augment b until we reach its upper bound. The program `gnfs` estimates the maximum value of $F_i(a,b)$ over all values of a and b for both polynomials. The polynomial for which this estimate is larger is sieved first. Probably fewer pairs (a,b) will have a smooth value of $F_i(a,b)$ for this polynomial, so fewer pairs have to be stored. Furthermore this largest value is used to decide upon the base of the logarithm, which we choose in such a way that the log of the maximum fits in one byte. Suppose we start sieving with polynomial f_j.

To sieve for the first polynomial f_j we fix b and initialize to zero an array that contains one byte per a-value. For every prime $p < B_j$ and every r with $f_j(r) \equiv 0 \bmod p$ we add $[\log p]$ (where $[\,\cdot\,]$ is the nearest integer function) to all array elements corresponding with $a \equiv br \bmod p$. For every prime $p < B_j$ with $p \mid c_{j,d_j}$ and $p \mid b$ we add $[\log p]$ to every array element. Then we split the a-interval recursively in subintervals until the value of $c_j(a,b) = F_j(a,b)/L_j^2$ does not vary more than a prescribed amount within a subinterval. If the value of an array element is close enough to $\log c_j(a,b)$, then $F_j(a,b)$ is potentially smooth and we store the value of a. Now the same sieving process takes place for the other polynomial f_{3-j}. If for a pair (a,b) both $F_1(a,b)$ and $F_2(a,b)$ are potentially smooth — (a,b) is now called a *candidate relation* —, we use trial division (where we first test if $a \equiv br \bmod p$ before applying an expensive multiple precision division of $F_i(a,b)$ on p) to extract all factors below B_i from $F_i(a,b)$, for $i = 1,2$. This is necessary, since during the sieving we use rounded logarithms and other techniques, which not only make the sieving faster, but also make the final value in the array elements less accurate.

(In [Golliver et al. 1994] experiments were made with repeating the sieving procedure once again, instead of using trial division. The candidate relations are marked in the sieving array. In a second sieving round the primes p themselves are stored instead of adding $[\log p]$ to the array elements for the candidate relations. Next the integers $F_1(a,b)$ and $F_2(a,b)$ are calculated for the candidate relations and the stored primes are divided out. This approach costs more memory, but is likely faster.)

By increasing p, and comparing the sieved logarithms with the sum of the logarithms of primes divided out of $F_i(a,b)$ during trial division so far, one can sometimes skip an interval of primes. If after the trial division there remains a composite part smaller than L_i^2, we try to factor it first using SQUFOF, and if that fails using Pollard Rho [Riesel 1985, pp. 191–198, 174–183]. A pair (a,b) is a relation if both $F_i(a,b)$ factor over the primes below B_i except for at most two large primes between B_i and L_i. It is stored together with the primes dividing $F_i(a,b)$ that exceed some user-determined *printing bounds* W_i, where $i = 1,2$. With these bounds W_i one can monitor the amount of output of the `gnfs` program. They should be chosen in such a way that it fits in the available disk space.

Using the lattice sieve, we only sieve over pairs (a,b) of which we know that one $F_i(a,b)$, say for $i = j \in \{1,2\}$, is divisible by a special large prime between $L^{(l)}$ and $L^{(u)}$, which are the user-chosen lower and upper bound for the large primes, respectively. The advantage is that the remaining part of $F_j(a,b)$ is more likely to be smooth. On the other hand we will miss the relations for which both $F_i(a,b)$ are smooth over the primes below $L^{(l)}$. For the implementation of the lattice sieve we use an extra feature implemented in the classical way of sieving. There we have a possibility of sieving over a sublattice of the (a,b)-pairs. We can choose an integral, nonsingular matrix $\boldsymbol{M}$ and sieve over pairs (a,b) of the form:

$$\begin{pmatrix} a \\ b \end{pmatrix} = \boldsymbol{M} \begin{pmatrix} x \\ y \end{pmatrix},$$

while the program sieves over x and y. This is done by substituting the expressions of a and b in terms of x and y in both $F_i(a,b)$ resulting in new polynomials $G_i(x,y)$, which are now the polynomials whose values should be smooth. Of course the roots of the polynomials F_i have to be adapted to the roots of the polynomials G_i. When a pair (x,y) is a relation, the corresponding pair (a,b), together with the primes dividing $G_i(x,y)$ and exceeding W_i, are stored.

The lattice sieve sieves for every prime q in the range $[L^{(l)}, L^{(u)}]$, for a fixed value of b over all roots $(r_1 : r_2) \in \{\mathcal{R}_1(q) \cup \mathcal{R}_2(q)\}$ with $r_2 \neq 0$. When sieving over a root $(r_1 : r_2)$ of $\mathcal{R}_j(q)$ we sieve only over the a-values with $a \equiv b r_1 r_2^{-1} \bmod q$, thus guaranteeing that $F_j(a,b)$ is divisible by q. This is the same as using a matrix

$$\boldsymbol{M} = \begin{pmatrix} q & r_1 r_2^{-1} \bmod q \\ 0 & 1 \end{pmatrix},$$

with y fixed to b and x in an interval such that $qx + b(r_1 r_2^{-1} \bmod q)$ just fits in the a-interval. (Note that, when we compare our notation with that used in [Pollard 1993b], we have $V_1 = (q,0)$ and $V_2 = (r_1 r_2^{-1} \bmod q, 1)$, and that we are applying the "sieving by rows" strategy.) $G_j(x,y)/q$ should be smooth over the primes below B_j, except for at most two large primes between B_j and q. The other $G_{3-j}(x,y)$ should be smooth over the primes below B_{3-j}, except for at most two large primes between B_{3-j} and q. Not allowing primes equal to or bigger than q to divide one of the $G_i(x,y)$ avoids generating duplicate relations, but misses relations having two large primes smaller than q for $G_j(x,y)$ and a large prime larger than q for $G_{3-j}(x,y)$. After we have sieved over all roots in $\mathcal{R}_1(q)$ and $\mathcal{R}_2(q)$ we take the next value of b; after we have sieved over all values of b we take the next prime in the interval $[L^{(u)}, L^{(l)}]$. We implemented lattice sieving only for the case of two quadratic polynomials.

Since the sieving is the most time-consuming step of the algorithm, its implementation is critical. It is a lot of work to sieve over a small prime p, and just a small amount of $[\log p]$ is added to the array elements. Therefore we sieve only over primes and prime powers larger than 30. Also we do not add $[\log p]$ to all array elements for primes $p < B_j$ with $p \mid c_{j,d_j}$ and $p \mid b$, but we divide $c_j(a,b)$ by p. Furthermore we split the a-interval into subintervals that fit in the secondary cache of the computer, making the sieving faster. For a group of small primes, which consists of the primes for which we sieve over a power rather than over the prime itself, we again split the subintervals into smaller subintervals which fit into the primary cache. The user can install several "early abort" bounds: if the leftover part of $F_i(a,b)$ after trial division over all primes below a bound $B < B_i$ is bigger than a user-specified constant times the square of the large prime bound, then the pair (a,b) is not considered to be a candidate for a relation and is thrown away. In the case of lattice sieving, the values of a with $a \equiv b r_1 r_2^{-1} \bmod q$ are far away from each other for a fixed value of b. In Section 5 we explained how we select polynomials such that we can increase the efficiency of the sieving by taking a huge a-interval and $b = 1$. Therefore we call it *line sieving*.

7. THE FILTERING

The aim of filtering is just the reduction of the amount of data. We want to find a subset $\mathcal{S}$ of all relations $\{(a,b)\}$ found in the sieving step and a subset $\mathcal{T}$ of the set of free rational primes p such that $\left(\prod_{\mathcal{T}} p\right)\left(\prod_{\mathcal{S}}(a - b\alpha_i)\right)$ is a square in $\mathbb{Q}_n[\alpha_i]$, for $i = 1, 2$. Therefore every algebraic prime p dividing one of the products $\left(\prod_{\mathcal{T}} p^{d_i}\right)\left(\prod_{\mathcal{S}} F_i(a,b)\right)$ for a certain root $(r_1 : r_2) \in \mathcal{R}_i(p)$ (from now on denoted by $p_{(r_1:r_2)}$) must occur to an even power with respect to this root. (Here we should see p^{d_i} as the product of one factor p for every root $(r_1 : r_2) \in \mathcal{R}_i(p)$). A prime $p_{(r_1:r_2)}$ occurs in a relation (a,b) for polynomial i if p divides $F_i(a,b)$ and $(a \bmod p : b \bmod p) = (r_1 : r_2)$. A prime $p_{(r_1:r_2)}$ occurs in a free relation for both polynomials if p is a free prime. We say that a prime $p_{(r_1:r_2)}$ occurs in a relation for polynomial i if it occurs in a relation (a,b) for polynomial i or if p is a free prime. It is obvious that a relation in which some prime $p_{(r_1:r_2)}$ occurs to an odd power for one of the two polynomials is useless, if this prime is not occurring in some other relation to an odd power for the same polynomial. The filtering stage throws away such relations. If a prime $p_{(r_1:r_2)}$ occurs to an odd power in just two relations for the same polynomial and one of them belongs to the set $\mathcal{S}$, the other one should also be part of $\mathcal{S}$. In the filtering stage the two relations are grouped into a relation-set. If one relation from a relation-set is chosen in the set $\mathcal{S}$, then all relations from that relation-set should be in $\mathcal{S}$. By creating the relation-set we have eliminated the need to take care of the prime $p_{(r_1:r_2)}$ when looking for the set $\mathcal{S}$. In this way the amount of data and the size of the matrix for the next linear algebra step are reduced.

The relations found in the sieving step are read in sequentially. In order to regulate the amount of memory used, the user first chooses a number of temporary files among which the data will be distributed. During the filtering process data from only one temporary file will be in the working memory of the computer. A hash function is implemented that distributes the primes equally over the temporary files. For all the primes in the input file with norm larger than some user-determined bound $U \geq \max(W_1, W_2)$ and occurring to an odd power in one of the $F_i(a, b)$, the `filter` program calculates the index of the corresponding temporary file by using the hash function on the prime. The relation is written to the file with the smallest number it gets from all these primes. We store a, b, and the primes that were written in the input file. Extra features in this program have been added, such as looking only at the primes below some user-determined bound and throwing away all the relations containing a prime bigger than some user-determined bound.

When all relations are read in and stored in the corresponding files, the combining and throwing out process starts. First all relations (a, b) of the first file are read in and stored in a heap [Standish 1980, § 3.7.1] in descending order according to the largest prime that led to the storage of the relation in this file. We start by considering the relations in which the largest prime p corresponding to this file occur. We calculate the root $(r_1 : r_2)$ for this prime for the relations (a, b) by calculating $(a \bmod p : b \bmod p)$. If $d_1 + d_2$ different roots for this prime appear in these relations, we append the free relation for this prime. If, while looking at a prime $p_{(r_1:r_2)}$, we see the same relation (a, b) twice, we throw out one of the occurrences. If some prime $p_{(r_1:r_2)}$ occurs exactly once for one of the polynomials, the corresponding relation is thrown out. If a prime $p_{(r_1:r_2)}$ occurs twice for one of the polynomials, the two corresponding relations are grouped in a relation-set. If the user wishes, for primes $p_{(r_1:r_2)}$ that occur just three times for one of the polynomials, the program replaces the three corresponding relations by two relation-sets of two relations each.

Next the resulting relation-sets and the relations that were not combined are stored at the next place corresponding to the hash function. If the relation or relation-set contains smaller primes to an odd power that correspond to the same file, we keep the relation(-set) in the working memory. We store the relation(-set) in the heap according to the largest of those primes. Otherwise, if the relation (-set) contains primes to an odd power that correspond to other files, we store the relation(-set) in the file among those with the smallest number exceeding the number of the file which we currently have in memory, or, if there is no such file, in the file among those with the smallest number. If the relation(-set) only contains larger primes to an odd power that correspond to the same file, we keep the relation(-set) in the same file, but write it to disk. If the relation(-set) does not contain any primes larger than W_i to an odd power anymore, we write it to an output file. This circular queue is constructed in such a way that, when trying to throw out relation(-set)s or combining relation(-set)s into (new) relation-sets for some prime p, all relation (-set)s containing that prime to an odd power will be considered. When we store a relation-set we store all relations it contains, together with their primes exceeding W_i and the free primes of the relation-set.

After the first temporary file is treated in this way, the same process takes place consecutively on the other files. Of course, relation-sets can also be combined with each other. Relation-sets that become too large, in the sense that they contain more relations than a user-determined bound, are thrown away, in an attempt to keep the matrix for the next linear algebra step sparse. The user can fix the maximum number of relation-sets that can be thrown away. When the last file has been processed the program starts again on the first file, until no changes have taken place in the last round or the number of passes has reached a user-chosen bound. Then all relation(-set)s that are still in the temporary files are written to the output file.

The `filter` program counts the number of relations and relation-sets that remain and the number of primes $p_{(r_1:r_2)}$ occurring to an odd power in one of the relations or relation-sets with norm larger than U. From these data one can estimate whether there are enough relations.

In practice we used `filter` several times for one number. To save disk space we chose big printing bounds, $W_1 = W_2 = 10^6$, say. First we applied the `filter` program to the output of the sieving step with $U = W_1$. On the remaining relation(-set)s we applied the program `factorrelations` to compute the prime factors between a smaller bound W' and W_i of $F_1(a,b)$ and $F_2(a,b)$ for all relations (a,b) and store the relation(-set)s, now with all primes exceeding W'. Then we again applied the `filter` program, now on all primes exceeding U', with $W' \le U' < U$. These steps were repeated until we reached a bound below which many primes occur at least four times, so no combining or throwing out could be done, or until we were content with the resulting matrix size.

Another method that can be used for reducing the amount of data is structured Gaussian elimination, described in [LaMacchia et al. 1991], for example. A comparison between our filtering method and structured Gaussian elimination has not yet been made.

8. THE BLOCK LANCZOS METHOD

After enough relations have been collected and the `filter` program has reduced the amount of data, we try to find a subset $\mathcal{S}$ of the remaining relations and a subset $\mathcal{T}$ of the set of free primes such that $\left(\prod_{\mathcal{T}} p\right)\left(\prod_{\mathcal{S}}(a - b\alpha_i)\right)$ is a square in $\mathbb{Q}_n[\alpha_i]$, for $i = 1, 2$. For simplicity, from now on we view a relation left after the filtering stage as a relation-set containing only one relation. To this collection of relation-sets we append a relation-set for every free prime below $\max(W_1, W_2)$, containing this prime.

A relation-set $\mathcal{V}$ consists of two (possibly empty) subsets $\mathcal{V}_f$ and $\mathcal{V}_r$ that contain the free primes and the relations of $\mathcal{V}$, respectively. For every relation-set $\mathcal{V}$ we construct a vector

$$\boldsymbol{v}(\mathcal{V}) = \sum_{\mathcal{V}_p} \boldsymbol{v}(p) + \sum_{\mathcal{V}_r} \boldsymbol{v}(a,b).$$

The vectors $\boldsymbol{v}(a,b)$ are as described in Sections 2 and 3; the vectors $\boldsymbol{v}(p)$ are described in Section 4. We build a matrix $\boldsymbol{M}$ whose columns are all vectors $\boldsymbol{v}(\mathcal{V})$. We remove the rows that contain only zeros. They correspond to primes $(q, (r_1 : r_2))$ occurring to an even power in every relation-set $\mathcal{V}$. We want to calculate some nontrivial vectors of the null space of this matrix.

Since Gaussian elimination [Knuth 1981, § 4.6.2, Algorithm N] requires too much memory for the large sparse matrices we have, we use a variation of the iterative Lanczos method. Proofs on both standard Lanczos and block Lanczos can be found in [Montgomery 1995]. The standard Lanczos algorithm starts with a symmetric, positive definite $k \times k$ matrix $\boldsymbol{A}$ over the field $K = \mathbb{R}$. If $\boldsymbol{b} \in \mathbb{R}^k$ we solve $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ by the following iterative procedure: set $\boldsymbol{w}_0 = \boldsymbol{b}$ and

$$\boldsymbol{w}_i = \boldsymbol{A}\boldsymbol{w}_{i-1} - \sum_{j=0}^{i-1} c_{ij}\boldsymbol{w}_j \tag{8.1}$$

for $i > 0$, where

$$c_{ij} = \frac{\boldsymbol{w}_j^T \boldsymbol{A}^2 \boldsymbol{w}_{i-1}}{\boldsymbol{w}_j^T \boldsymbol{A}\boldsymbol{w}_j}.$$

It can be shown that after at most k iterations we will find $\boldsymbol{w}_i = \mathbf{0}$. If l is the first value of i such that $\boldsymbol{w}_i = \mathbf{0}$, we have $\boldsymbol{w}_i^T \boldsymbol{A}\boldsymbol{w}_i \neq 0$ for $0 \leq i < l$, $\boldsymbol{w}_j^T \boldsymbol{A}\boldsymbol{w}_i = 0$ for $i \neq j$, and $\boldsymbol{A}\mathcal{W} \subseteq \mathcal{W}$, where $\mathcal{W}$ is the span of $\boldsymbol{w}_0, \boldsymbol{w}_1, \ldots, \boldsymbol{w}_{l-1}$. One can deduce that

$$\boldsymbol{x} = \sum_{i=0}^{l-1} \frac{\boldsymbol{w}_i^T \boldsymbol{b}}{\boldsymbol{w}_i^T \boldsymbol{A}\boldsymbol{w}_i} \boldsymbol{w}_i$$

is a solution of $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. Since $\boldsymbol{w}_j^T \boldsymbol{A}^2 \boldsymbol{w}_{i-1} = 0$ for $j < i - 2$ we can simplify the calculation of $\boldsymbol{w}_i$ to

$$\boldsymbol{w}_i = \boldsymbol{A}\boldsymbol{w}_{i-1} - c_{i,i-1}\boldsymbol{w}_{i-1} - c_{i,i-2}\boldsymbol{w}_{i-2} \tag{8.2}$$

for $i \geq 2$.

Standard Lanczos can also be applied over other fields, provided that $\boldsymbol{w}_i^T \boldsymbol{A}\boldsymbol{w}_i \neq 0$ when $\boldsymbol{w}_i \neq \mathbf{0}$ during this process. Working over the field $\mathbb{F}_2$ instead of $\mathbb{R}$ has the advantage that one can apply a matrix to N different vectors simultaneously, where N is the computer word size. Inspired by work of Coppersmith [1993], Montgomery [1995] implemented the block Lanczos method, which exploits this advantage.

The block Lanczos algorithm creates a sequence of subspaces $\mathcal{W}_i$ instead of vectors $\boldsymbol{w}_i$. Applying standard Lanczos over $\mathbb{F}_2$ has the problem that in approximately half of the cases the requirement $\boldsymbol{w}_i^T \boldsymbol{A}\boldsymbol{w}_i \neq 0$ if $\boldsymbol{w}_i \neq \mathbf{0}$ is violated. In the block Lanczos algorithm the analogous requirement is that no nonzero vector in $\mathcal{W}_i$ is $\boldsymbol{A}$-orthogonal to $\mathcal{W}_i$. This will hold when $\boldsymbol{W}_i^T \boldsymbol{A}\boldsymbol{W}_i$ is invertible, where $\boldsymbol{W}_i$ is a matrix whose column vectors span $\mathcal{W}_i$.

For $\boldsymbol{A}$ a symmetric $k \times k$ matrix over a field K and $\boldsymbol{V}_0$ an arbitrary $k \times N$ matrix, the block Lanczos algorithm proceeds by setting, for $i = 0, 1, \ldots$,

$$\boldsymbol{W}_i = \boldsymbol{V}_i S_i \tag{8.3}$$

$$\boldsymbol{C}_{i+1,j} = (\boldsymbol{W}_j^T \boldsymbol{A}\boldsymbol{W}_j)^{-1} \boldsymbol{W}_j^T \boldsymbol{A}(\boldsymbol{A}\boldsymbol{W}_i \boldsymbol{S}_i^T + \boldsymbol{V}_i), \tag{8.4}$$

$$\boldsymbol{V}_{i+1} = \boldsymbol{A}\boldsymbol{W}_i \boldsymbol{S}_i^T + \boldsymbol{V}_i - \sum_{j=0}^{i} \boldsymbol{W}_j C_{i+1,j}. \tag{8.5}$$

In (8.4), j ranges from 0 through i. In (8.3) the $N \times N_i$ matrix $\boldsymbol{S}_i$ (where $N_i \leq N$) consists of zeros except for exactly one 1 per column and at most one 1 per row, thus selecting columns from $\boldsymbol{V}_i$ for $\boldsymbol{W}_i$. We choose the columns of $\boldsymbol{V}_i$ in such a way that the corresponding columns of $\boldsymbol{V}_i^T \boldsymbol{A}\boldsymbol{V}_i$ are a linearly independent spanning set of all columns of $\boldsymbol{V}_i^T \boldsymbol{A}\boldsymbol{V}_i$. Thus $N_i = \operatorname{rank}(\boldsymbol{V}_i^T \boldsymbol{A}\boldsymbol{V}_i)$, and one can prove that the resulting matrix $\boldsymbol{W}_i^T \boldsymbol{A}\boldsymbol{W}_i$ is invertible. The iteration process stops when $\boldsymbol{V}_i^T \boldsymbol{A}\boldsymbol{V}_i = \mathbf{0}$, for $i = l$ say.

If $\boldsymbol{V}_l = \mathbf{0}$, the matrix $\boldsymbol{W}_i^T \boldsymbol{A}\boldsymbol{W}_i$ is invertible for $0 \leq i < l$, and we have

$$\boldsymbol{W}_j^T \boldsymbol{A}\boldsymbol{W}_i = \mathbf{0} \quad \text{for } i \neq j, \tag{8.6}$$

$$\boldsymbol{W}_j^T \boldsymbol{A}\boldsymbol{V}_i = \mathbf{0} \quad \text{for } 0 \leq j < i \leq l, \tag{8.7}$$

and finally $\boldsymbol{A}\mathcal{W} \subseteq \mathcal{W}$, where $\mathcal{W}$ is the span of $\mathcal{W}_0, \mathcal{W}_1, \ldots, \mathcal{W}_{l-1}$. If $\boldsymbol{b} \in \mathcal{W}$ one can further deduce that the vector

$$\boldsymbol{x} = \sum_{i=0}^{l-1} \boldsymbol{W}_i (\boldsymbol{W}_i^T \boldsymbol{A}\boldsymbol{W}_i)^{-1} \boldsymbol{W}_i^T \boldsymbol{b}$$

is a solution of $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$.

If we can choose $\boldsymbol{S}_i$ in such a way that $\operatorname{span}(\boldsymbol{V}_i) \subseteq \operatorname{span}(\mathcal{W}_0, \mathcal{W}_1, \ldots, \mathcal{W}_{i+1})$, it is possible to simplify the calculation of $\boldsymbol{V}_{i+1}$ in (8.5) in a way similar to that in which the calculation of $\boldsymbol{w}_i$ in (8.1) was simplified to (8.2):

$$\begin{aligned} \boldsymbol{V}_{i+1} = \boldsymbol{A}\boldsymbol{W}_i \boldsymbol{S}_i^T + \boldsymbol{V}_i - \boldsymbol{W}_i \boldsymbol{C}_{i+1,i} \\ - \boldsymbol{W}_{i-1}\boldsymbol{C}_{i+1,i-1} - \boldsymbol{W}_{i-2}\boldsymbol{C}_{i+1,i-2}, \end{aligned} \tag{8.8}$$

for $i \geq 2$. The requirement is fulfilled when the columns of $\boldsymbol{V}_i$ are in $\operatorname{span}(\mathcal{W}_i, \mathcal{W}_{i+1})$. From (8.8) we can deduce that

$$\boldsymbol{V}_{i+1} = \boldsymbol{A}\boldsymbol{W}_i \boldsymbol{S}_i^T + \boldsymbol{V}_i - \boldsymbol{W},$$

where $\boldsymbol{W}$ is a $k \times N$ matrix whose columns are linear combinations of the columns of $\boldsymbol{W}_i$, $\boldsymbol{W}_{i-1}$ and $\boldsymbol{W}_{i-2}$. Notice that the columns that were not selected from $\boldsymbol{V}_i$ are zero in $\boldsymbol{S}_i^T$ and in $\boldsymbol{A}\boldsymbol{W}_i \boldsymbol{S}_i^T$ as well. Therefore a nonselected column of $\boldsymbol{V}_i$ is equal to the sum of the corresponding columns of

$\boldsymbol{V}_{i+1}$ and $\boldsymbol{W}$. Using (8.7) and (8.6) we can deduce that such a column of $\boldsymbol{W}$ must be a linear combination of the columns of $\boldsymbol{W}_i$ only. Choosing the columns of $\boldsymbol{V}_{i+1}$ which were not selected in $\boldsymbol{V}_i$ guarantees that the nonselected columns of $\boldsymbol{V}_i$ are in $\mathrm{span}(\mathcal{W}_i, \mathcal{W}_{i+1})$. If these columns of $\boldsymbol{V}_i$ are independent but the corresponding columns of $\boldsymbol{V}_{i+1}^T \boldsymbol{A} \boldsymbol{V}_{i+1}$ are dependent we cannot fulfill the requirement that $\boldsymbol{W}_{i+1}^T \boldsymbol{A} \boldsymbol{W}_{i+1}$ is invertible, and the algorithm fails. In practice this has never happened. Otherwise we choose a spanning set of columns for $\boldsymbol{V}_{i+1}^T \boldsymbol{A} \boldsymbol{V}_{i+1}$ including the columns that were not selected for $\boldsymbol{V}_i$, and choose $\boldsymbol{S}_{i+1}$ accordingly.

To apply the block Lanczos method to our matrix $\boldsymbol{M}$, we have to deal with several obstructions. First $\boldsymbol{M}$ need not be symmetric and therefore we apply the algorithm to the symmetric matrix $\boldsymbol{A} = \boldsymbol{M}^T \boldsymbol{M}$. It is obvious that any solution of $\boldsymbol{M}\boldsymbol{x} = \boldsymbol{0}$ satisfies $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}$, but the converse need not be true. Secondly, if we want to find a vector from the null space of $\boldsymbol{A}$ and start with $\boldsymbol{b} = \boldsymbol{0}$, we will find the trivial solution. We overcome this problem by starting with a random vector $\boldsymbol{y}$ and taking $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{y}$. When $\boldsymbol{x}$ is a solution of $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b} = \boldsymbol{A}\boldsymbol{y}$, then $\boldsymbol{x} - \boldsymbol{y}$ will be a random vector from the null space of $\boldsymbol{A}$. Thirdly, several vectors from the null space of $\boldsymbol{M}$ have to be found, since not every dependency corresponding with such a vector will lead to a nontrivial factor of n. Note that during the iteration steps the vector $\boldsymbol{b}$ only is involved in the calculation of the solution vector $\boldsymbol{x}$. Therefore replacing $\boldsymbol{b}$ by a $k \times N$ matrix $\boldsymbol{B}$ in the calculation of $\boldsymbol{x}$ will give a solution of $\boldsymbol{A}\boldsymbol{X} = \boldsymbol{B}$, with $\boldsymbol{X}$ also a $k \times N$ matrix. To find several vectors in the null space of $\boldsymbol{A}$ we start with a random $k \times N$ matrix $\boldsymbol{Y}$ and calculate solutions of $\boldsymbol{A}\boldsymbol{X} = \boldsymbol{A}\boldsymbol{Y}$. The N column vectors of $\boldsymbol{X} - \boldsymbol{Y}$ will be random vectors in the null space of $\boldsymbol{A}$ and we extract the ones which are also in the null space of $\boldsymbol{M}$.

Final obstructions are the two requirements for $\boldsymbol{x}$ to be a solution of $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$. First $\boldsymbol{b}$ has to be in $\mathcal{W} = \mathrm{span}(\mathcal{W}_0, \mathcal{W}_1, \ldots, \mathcal{W}_{l-1})$. This can be arranged by initializing $\boldsymbol{V}_0$ as $\boldsymbol{A}\boldsymbol{Y}$, where $\boldsymbol{Y}$ is a random $k \times N$ matrix. Secondly, the algorithm often terminates with $\boldsymbol{V}_l^T \boldsymbol{A} \boldsymbol{V}_l = \boldsymbol{0}$ but $\boldsymbol{V}_l \neq \boldsymbol{0}$. Montgomery presumes that the column vectors of $\boldsymbol{A}(\boldsymbol{X} - \boldsymbol{Y})$ and $\boldsymbol{A}\boldsymbol{V}_l$ are both in $\mathrm{span}(\boldsymbol{V}_l)$, which has maximal rank N, but in practice the rank is much smaller. We may expect that some linear combinations of these vectors are in the null space of $\boldsymbol{A}$. Combined with the need to find vectors in the null space of $\boldsymbol{M}$ instead of $\boldsymbol{A}$, it suffices to construct a suitable matrix $\boldsymbol{U}$ such that $\boldsymbol{M}\boldsymbol{Z}\boldsymbol{U} = \boldsymbol{0}$, where $\boldsymbol{Z}$ is a $k \times 2N$ matrix of the columns of $\boldsymbol{X} - \boldsymbol{Y}$ and $\boldsymbol{V}_l$. We first compute $\boldsymbol{M}\boldsymbol{Z}$. Then we determine a matrix $\boldsymbol{U}$ whose columns span the null space of $\boldsymbol{M}\boldsymbol{Z}$. The output is a basis for $\boldsymbol{Z}\boldsymbol{U}$.

For implementing the calculation of $\boldsymbol{V}_{i+1}$ in (8.8) one can bring further down the number of calculations by using the following steps: for $i = 0, 1, \ldots$, set

$$\boldsymbol{V}_{i+1} = \boldsymbol{A}\boldsymbol{V}_i\boldsymbol{S}_i\boldsymbol{S}_i^T + \boldsymbol{V}_i\boldsymbol{D}_{i+1} + \boldsymbol{V}_{i-1}\boldsymbol{E}_{i+1} + \boldsymbol{V}_{i-2}\boldsymbol{F}_{i+1},$$

where

$$\begin{aligned}
\boldsymbol{D}_{i+1} &= \boldsymbol{I}_N - \boldsymbol{W}_i^*(\boldsymbol{V}_i^T\boldsymbol{A}^2\boldsymbol{V}_i\boldsymbol{S}_i\boldsymbol{S}_i^T + \boldsymbol{V}_i^T\boldsymbol{A}\boldsymbol{V}_i),\\
\boldsymbol{W}_i^* &= \boldsymbol{S}_i(\boldsymbol{S}_i^T\boldsymbol{V}_i^T\boldsymbol{A}\boldsymbol{V}_i\boldsymbol{S}_i)^{-1}\boldsymbol{S}_i^T,\\
\boldsymbol{E}_{i+1} &= -\boldsymbol{W}_{i-1}^*\boldsymbol{V}_i^T\boldsymbol{A}\boldsymbol{V}_i\boldsymbol{S}_i\boldsymbol{S}_i^T,\\
\boldsymbol{F}_{i+1} &= -\boldsymbol{W}_{i-2}^*(\boldsymbol{I}_N - \boldsymbol{V}_{i-1}^T\boldsymbol{A}\boldsymbol{V}_{i-1}\boldsymbol{W}_{i-1}^*)\\
&\qquad(\boldsymbol{V}_{i-1}^T\boldsymbol{A}^2\boldsymbol{V}_{i-1}\boldsymbol{S}_{i-1}\boldsymbol{S}_{i-1}^T + \boldsymbol{V}_{i-1}^T\boldsymbol{A}\boldsymbol{V}_{i-1})\boldsymbol{S}_i\boldsymbol{S}_i^T,
\end{aligned}$$

and where, for $i < 0$, $\boldsymbol{W}_i^*$ and $\boldsymbol{V}_i$ are $\boldsymbol{0}$ and $\boldsymbol{S}_i$ is $\boldsymbol{I}_N$. When $\boldsymbol{S}_{i-1} = \boldsymbol{I}_N$ then $\boldsymbol{F}_{i+1} = \boldsymbol{0}$ and the term $\boldsymbol{V}_{i-2}\boldsymbol{F}_{i+1}$ in the expression for $\boldsymbol{V}_{i+1}$ can be omitted.

For large sparse matrices Lanczos' algorithm requires less storage than Gaussian elimination. It only needs the original matrix and some extra vectors of length k and some $N \times N$ matrices, while Gaussian elimination causes fill-in and therefore needs approximately k^2 bits. When $\boldsymbol{M}$ has d nonzero entries per row on average, the time needed by block Lanczos is $\mathcal{O}(dk^2/N) + \mathcal{O}(k^2)$. When d is much smaller than k this is considerably better than $\mathcal{O}(k^3/N)$ for Gaussian elimination.

In practice we make $\boldsymbol{M}$ extra sparse by removing the first row containing only ones and not appending any character rows. Also one could implement the possibility to remove some of the dense rows corresponding to small primes. If $\boldsymbol{M}$ is a $k_1 \times k_2$ matrix, the output of the block Lanczos algorithm will consist of a $k_2 \times N$ matrix $\boldsymbol{P}$ with N "pseudo-dependencies" of which we still have to find linear combinations to get a set $\mathcal{S}$ we look for. We solve this problem here, although in our implementation it is a part of the square root program. For several quadratic characters $(q, (s_1 : s_2))$ — chosen as described in Section 2 with q larger than any prime dividing any $F_i(a, b)$ — we form a vector $\boldsymbol{q}_{(q,(s_1:s_2))}$ of length k_2 by inserting a zero for all of the k_2 relation-sets $\mathcal{V}$ with

$$\prod_{\mathcal{V}_p}\Bigl(\frac{p}{q}\Bigr)\prod_{\mathcal{V}_f}\Bigl(\frac{a-b\ (s_1s_2^{-1} \bmod q)}{q}\Bigr)=1$$

and a one otherwise. A vector of length N orthogonal to all vectors $\boldsymbol{q}_{(q,(s_1:s_2))}\boldsymbol{P}$ is indicating a linear combination of the N pseudo-dependencies which is favourable to all chosen quadratic characters. We construct a basis for the space orthogonal to all vectors $\boldsymbol{q}_{(q,(s_1:s_2))}\boldsymbol{P}$. Each of these basis vectors indicates which pseudo-dependencies of $\boldsymbol{P}$ should be combined for a real dependency, thereby indicating a set $\mathcal{S}$.

9. EXTRACTING THE SQUARE ROOT

At this stage we have two squares $\beta^2 = \bigl(\prod_{p\in\mathcal{T}} p\bigr) \times \bigl(\prod_{(a,b)\in\mathcal{S}}(a-b\alpha_1)\bigr)$ and $\gamma^2 = \bigl(\prod_{p\in\mathcal{T}} p\bigr)\bigl(\prod_{(a,b)\in\mathcal{S}}(a-b\alpha_2)\bigr)$ in $\mathbb{Q}_n[\alpha_1]$ and $\mathbb{Q}_n[\alpha_2]$, respectively. We have to calculate β and γ. If we write both squares as polynomials of degree less than d_i in α_i, the coefficients will be gigantic. Then a conventional method such as the one described in [Cohen 1993, § 3.6.2] cannot be used. Couveignes [1993] calculates the square roots modulo several primes and applies the Chinese Remainder Theorem, a method that presently works only for number fields of odd degree.

Montgomery [1994] attacks the problem using an iterative process. He starts by partitioning the set $\mathcal{S}$ in two subsets $\mathcal{S}_1$ and $\mathcal{S}_2$ and the set $\mathcal{T}$ in two subsets $\mathcal{T}_1$ and $\mathcal{T}_2$ to advance the cancellation of primes $p_{(r_1:r_2)}$ in both products

$$\frac{\prod_{p\in\mathcal{T}_1} p^{d_i}\prod_{(a,b)\in\mathcal{S}_1} F_i(a,b)}{\prod_{p\in\mathcal{T}_2} p^{d_i}\prod_{(a,b)\in\mathcal{S}_2} F_i(a,b)}, \tag{9.1}$$

$i = 1, 2$. Here, the expressions p^{d_i} for the free primes $p \in \{\mathcal{T}_1 \cup \mathcal{T}_2\}$ should be seen as the product of one factor p for every root $(r_1 : r_2) \in \mathcal{R}_i(p)$. We can for example choose $\mathcal{S}_1 = \mathcal{S}$, $\mathcal{S}_2$ empty, $\mathcal{T}_1 = \mathcal{T}$ and $\mathcal{T}_2$ empty; or we can distribute $\mathcal{S}$ and $\mathcal{T}$ over the sets $\mathcal{S}_j$ and $\mathcal{T}_j$ randomly. At the end of this section we will see how we tried to optimize this selection.

Set

$$\nu_1^2 = \frac{\prod_{\mathcal{T}_1} p\prod_{\mathcal{S}_1}(a-b\alpha_1)}{\prod_{\mathcal{T}_2} p\prod_{\mathcal{S}_2}(a-b\alpha_1)},$$
$$\nu_2^2 = \frac{\prod_{\mathcal{T}_1} p\prod_{\mathcal{S}_1}(a-b\alpha_2)}{\prod_{\mathcal{T}_2} p\prod_{\mathcal{S}_2}(a-b\alpha_2)};$$

then we will calculate ν_1 and ν_2, for which the congruence $(\varphi_1(\nu_1))^2 \equiv (\varphi_2(\nu_2))^2 \bmod n$ holds. The following algorithm is applied twice, first to calculate $\nu = \nu_1$ and then to calculate $\nu = \nu_2$. In the rest of this section we suppress the index i when referring to ν_i, f_i, d_i, $c_{i,k}$ and α_i.

Starting with $\tau_1 = \nu^2$, where ν is unknown, we will approximate in iteration step $j \geq 1$ the numerator (if j is odd) or the denominator (if j is even) of $\sqrt{\tau_j}$ by η_j (to be explained below) and calculate τ_{j+1} using the formula

$$\tau_{j+1} = \tau_j \cdot (\eta_j^2)^{(-1)^j}. \tag{9.2}$$

Hence

$$\nu^2 = \tau_{j+1}\frac{\prod_{l=1}^{\lfloor\frac{j+1}{2}\rfloor}\eta_{2l-1}^2}{\prod_{l=1}^{\lfloor\frac{j}{2}\rfloor}\eta_{2l}^2}. \tag{9.3}$$

The product of the norms of the numerator and the denominator of τ_{j+1} in (9.3) will decrease at every iteration step. Small norms of numerator and denominator, however, do not guarantee that the coefficients of τ_{j+1} as a polynomial of degree $\leq d-1$ in α are small. Let $\alpha_1, \alpha_2, \ldots, \alpha_d$ be the conjugates of α. For any polynomial $h(x) \in \mathbb{Q}[x]$

of degree at most $d-1$ the Lagrange interpolation formula gives

$$h(x) = \sum_{k=0}^{d-1} x^k \sum_{l=1}^{d} h(\alpha_l) c_{kl},$$

where the c_{kl} can be calculated from

$$\frac{f(x)}{(x-\alpha_l)f'(\alpha_l)} = \sum_{k=0}^{d-1} c_{kl} x^k.$$

Therefore we can bound the coefficients of $h(x)$ in terms of the $|h(\alpha_l)|$. We use this observation by choosing the approximation η_j in such a way that not only the product of the numerator and the denominator norms of the successive τ_{j+1}'s decreases, but also $|\tau_{j+1}(\alpha_l)|$ tends to decrease for all l. When the norms and the embeddings become small enough, we will express the final τ_{j+1} as a polynomial of degree $\leq d-1$ in α and find its square root by using the computer package PARI [Batut et al. 1995].

In order to find the η_j's we work with ideals. Denote by $\mathcal{O}$ the ring of integers in $\mathbb{Q}[\alpha]$ and for $x_j \in \mathbb{Q}[\alpha]$ by $\langle x_1, \ldots, x_n\rangle\mathcal{O}$ the fractional ideal generated by $x_1, \ldots, x_n$ in $\mathcal{O}$. Suppose

$$\langle\sqrt{\tau_j}\rangle\mathcal{O} = \frac{\prod_{l\in\mathcal{L}_{j1}} \mathcal{P}_l^{c_l}}{\prod_{l\in\mathcal{L}_{j2}} \mathcal{P}_l^{c_l}} \tag{9.4}$$

is the factorization of $\langle\sqrt{\tau_j}\rangle\mathcal{O}$ into prime ideals $\mathcal{P}_l$ of $\mathcal{O}$, where $c_l \in \mathbb{Z}_+$ for all l. At each iteration step we select an ideal $\mathcal{I}$ dividing the numerator (if j is odd) or the denominator (if j is even) of (9.4). The approximation η_j will be a 'small' element of $\mathcal{I}$. If $\mathcal{I}$ divides the numerator, we divide $\langle\sqrt{\tau_j}\rangle\mathcal{O}$ by $\langle\eta_j\rangle\mathcal{O}$ and this will result in the disappearance of $\mathcal{I}$ in the numerator of $\langle\sqrt{\tau_j}\rangle\mathcal{O}$ and the appearance of a new integral ideal $\mathcal{Q}$ of $\mathcal{O}$ in the denominator of $\langle\sqrt{\tau_j}\rangle\mathcal{O}$. If $\mathcal{I}$ divides the denominator the converse happens. If $N(\mathcal{I})$ is sufficiently large, than $N(\mathcal{Q})$ will be much smaller than $N(\mathcal{I})$. In this way the product of the numerator and the denominator norms will decrease every iteration step.

To factor $\langle\tau_j\rangle\mathcal{O}$ into prime ideals, we use the ideal

$$\mathcal{J} = \langle c_d,\ c_d\alpha + c_{d-1},\ c_d\alpha^2 + c_{d-1}\alpha + c_{d-2},\ \ldots,\ c_d\alpha^{d-1} + c_{d-1}\alpha^{d-2} + \cdots + c_1\rangle\mathcal{O}.$$

We have $\mathcal{J} \subseteq \mathcal{O}$ and $\langle 1, \alpha\rangle\mathcal{O}\mathcal{J} = \mathcal{O}$ [Montgomery 1994]. From this we deduce $\langle a-b\alpha\rangle\mathcal{O}\mathcal{J} \subseteq \mathcal{O}$. Therefore, if we multiply an ideal $\langle a-b\alpha\rangle\mathcal{O}$ by $\mathcal{J}$ we can factor the result in prime ideals, all with a positive exponent. The ideals $\langle p\rangle\mathcal{O}$ with $p \in \mathcal{T}$ are already integral, so multiplication with $\mathcal{J}$ is not necessary for these ideals. We start with

$$\langle\tau_1\rangle\mathcal{O} = \frac{\mathcal{J}^{\#\mathcal{S}_2} \prod_{\mathcal{T}_1}\langle p\rangle\mathcal{O} \prod_{\mathcal{S}_1}\{\langle a-b\alpha\rangle\mathcal{O}\mathcal{J}\}}{\mathcal{J}^{\#\mathcal{S}_1} \prod_{\mathcal{T}_2}\langle p\rangle\mathcal{O} \prod_{\mathcal{S}_2}\{\langle a-b\alpha\rangle\mathcal{O}\mathcal{J}\}} \tag{9.5}$$

and factor the ideals $\langle p\rangle\mathcal{O}$ and $\langle a-b\alpha\rangle\mathcal{O}\mathcal{J}$ into prime ideals. Therefore we split the primes in two subsets: the set of "special" primes which divide the index $[\mathcal{O} : \mathbb{Z}[\alpha]]$ and the remaining primes which we call normal. To every prime p and every root $(r_1 : r_2) \in \mathcal{R}(p)$ there correspond prime ideals dividing $\langle p\rangle\mathcal{O}$ if p is an element of $\mathcal{T}_1 \cup \mathcal{T}_2$ or dividing $\langle a-b\alpha\rangle\mathcal{O}\mathcal{J}$ if p divides $F(a,b)$ ($(a,b) \in \mathcal{S}_1 \cup \mathcal{S}_2$). For a special prime there may exist more prime ideals corresponding to the same root, but for a normal prime p the prime ideal $\mathcal{P}$ corresponding to a root $(r_1 : r_2) \in \mathcal{R}(p)$ is uniquely determined. Based on practical experience Montgomery suspects — which we cannot prove — that in the latter case the correspondence is given by

$$\mathcal{P} = \begin{cases} \langle p, c_d\alpha - c_d r_1 r_2^{-1} \bmod p\rangle\mathcal{O} & \text{if } p \nmid c_d, \\ \langle p\rangle\mathcal{O} + \mathcal{J} & \text{if } p \mid c_d,\ r_2 = 0, \\ \mathcal{J}\cdot\langle p, \alpha - r_1 r_2^{-1} \bmod p\rangle\mathcal{O} & \text{if } p \mid c_d,\ r_2 \neq 0. \end{cases} \tag{9.6}$$

For the special primes p and for all their roots $(r_1 : r_2) \in \mathcal{R}(p)$, we calculate the ideal $\mathcal{P}$ using (9.6) and factor it into prime ideals with help of the computer package PARI. While we read in the free primes and relations we accumulate a product of the factors of the right hand side of (9.5). We make a hash table containing an entry for each normal prime p and $(r_1 : r_2) \in \mathcal{R}(p)$ we encounter. Each entry contains the exponent of the corresponding prime ideal in the accumulated product so far: if we meet a normal prime $p_{(r_1:r_2)}$ dividing a free prime or $F(a,b)$ in the numerator (or denominator) of $\langle\tau_1\rangle\mathcal{O}$ to the power x, we add (or subtract) x to this exponent. If we encounter a special prime $p_{(r_1:r_2)}$, dividing a free prime p or one of the $F(a,b)$, we use PARI to calculate the valuation of $\langle p\rangle\mathcal{O}$ or $\langle a-b\alpha\rangle\mathcal{O}\mathcal{J}$, respectively, at the ideals of $p_{(r_1:r_2)}$ we computed earlier. Also for these special ideals we keep track of the exponent of the ideal in the accumulated product so far. We also have to keep track of the exponent of $\mathcal{J}$ in the accumulated product. For each ideal $\langle a-b\alpha\rangle\mathcal{O}\mathcal{J}$ in the numerator or denominator we add or subtract 1, respectively. When we have read in all free primes and relations we have factored $\langle\tau_1\rangle\mathcal{O}$ into prime ideals and a power of $\mathcal{J}$.

Now we start the iterative approximation process in which we use the LLL lattice basis reduction algorithm [Lenstra et al. 1982]. Assume we want to simplify the numerator. The algorithm selects an ideal $\mathcal{I}=\mathcal{P}_1^{s_1}\dots\mathcal{P}_k^{s_k}$ dividing the numerator of $\langle\sqrt{\tau_j}\rangle\mathcal{O}$, with $s_r>0$ for all r. $N(\mathcal{I})$ should be chosen as large as computationally convenient for this first lattice basis reduction. Let $\{a_1,\dots,a_d\}$ be an integral bases of $\mathcal{O}$. With help of PARI we construct a basis in Hermite Normal Form (HNF) [Cohen 1993, § 4.7.2] expressed in $\{a_1,\dots,a_d\}$ for each prime ideal $\mathcal{P}_l$ occurring in $\mathcal{I}$. PARI uses these bases to construct a basis of $\mathcal{I}$, also in HNF expressed in $\{a_1,\dots,a_d\}$. Then we apply a lattice basis reduction to these d basis vectors of $\mathcal{I}$. We find a basis of $\mathcal{I}$ consisting of 'small' vectors. In practice, when using one of these small vectors for our approximation η_j, the norm of the numerator of $\langle\tau_{j+1}\rangle\mathcal{O}$ will decrease by a factor $N(\mathcal{I})$ in comparison with the norm of the numerator of $\langle\tau_j\rangle\mathcal{O}$. In comparison with the norm of the denominator of $\langle\tau_j\rangle\mathcal{O}$, the norm of the denominator of $\langle\tau_{j+1}\rangle\mathcal{O}$ will increase by a factor much smaller than $N(\mathcal{I})$.

We apply a second lattice basis reduction to a slight modification of the basis which we find after the first lattice basis reduction, to search for an element η_j in $\mathcal{I}$, which still has the same effect on the norm of $\langle\tau_{j+1}\rangle\mathcal{O}$, but yields small $|\tau_{j+1}(\alpha_l)|$ for all l. Let $\boldsymbol{v}^{(1)}$, $\boldsymbol{v}^{(2)}$, …, $\boldsymbol{v}^{(d)}$ be the reduced basis after the first lattice basis reduction, where $\boldsymbol{v}^{(r)}=\sum_{k=0}^{d-1}v_k^{(r)}\alpha^k$. While we read in the free primes and relations we calculate an approximation of $\tau_1(\alpha_l)$ for $1\le l\le d$. We choose c such that

$$c^d=\frac{L_{\max}\ (N(\tau_j))^{1/2}}{N(\mathcal{I})\ (\mathrm{Disc}(f/c_d))^{1/2}},$$

where $\mathrm{Disc}(g)$ is the discriminant of the polynomial g and $L_{\max}=10^{100}$. If all conjugates $\alpha_1,\dots,\alpha_d$ of α are real, we construct the vectors

$$\boldsymbol{T}\boldsymbol{v}^{(r)}=\Bigl(v_0^{(r)},v_1^{(r)},\dots,v_{d-1}^{(r)},\ c\frac{\boldsymbol{v}^{(r)}(\alpha_1)}{|\tau_j(\alpha_1)|^{1/2}},\dots,c\frac{\boldsymbol{v}^{(r)}(\alpha_d)}{|\tau_j(\alpha_d)|^{1/2}}\Bigr)^T$$

for $1\le r\le d$. If α_s and α_t are complex conjugates, we replace

$$c\frac{\boldsymbol{v}^{(r)}(\alpha_s)}{|\tau_j(\alpha_s)|^{1/2}}\quad\text{by}\quad c\sqrt{2}\frac{\mathrm{Re}(\boldsymbol{v}^{(r)}(\alpha_s))}{|\tau_j(\alpha_s)|^{1/2}}$$

and

$$c\frac{\boldsymbol{v}^{(r)}(\alpha_t)}{|\tau_j(\alpha_t)|^{1/2}}\quad\text{by}\quad c\sqrt{2}\frac{\mathrm{Im}(\boldsymbol{v}^{(r)}(\alpha_s))}{|\tau_j(\alpha_s)|^{1/2}}$$

in the construction of $\boldsymbol{T}\boldsymbol{v}^{(r)}$. In this way all entries of the $\boldsymbol{T}\boldsymbol{v}^{(r)}$ will be real and the absolute value of the determinant of the matrix formed by the last d entries of these vectors remains the same. The determinant equals $\pm L_{\max}$, which constant has been chosen in such a way that the second lattice basis reduction algorithm performs well. We apply the second lattice basis reduction to the vectors $\{\boldsymbol{T}\boldsymbol{v}^{(r)}\}_{r=1}^d$ and take the first d coordinates of one of the resulting vectors for η_j. When dividing τ_j by η_j^2 the ideal $\mathcal{I}$ in the numerator of $\langle\tau_j\rangle\mathcal{O}$ will disappear. At the same time the denominator of $\langle\tau_j\rangle\mathcal{O}$ will be multiplied by the square of a new ideal

$$\langle\eta_j\rangle\mathcal{O}/\mathcal{I}=:\mathcal{Q}. \tag{9.7}$$

In practice also for this η_j we have that $N(\mathcal{Q})$ is much smaller than $N(\mathcal{I})$.

If we simplify the denominator of τ_j we select an ideal $\mathcal{I} = \mathcal{P}_1^{s_1} \dots \mathcal{P}_k^{s_k}$ which divides the denominator of $\langle\sqrt{\tau_j}\rangle\mathcal{O}$. For the first LLL reduction the algorithm proceeds as above. For the second LLL reduction the vectors $\boldsymbol{T}\boldsymbol{v}^{(r)}$ become

$$\boldsymbol{T}\boldsymbol{v}^{(r)} = \Bigl(v_0^{(r)}, \dots, v_{d-1}^{(r)}, \\ c\,\boldsymbol{v}^{(r)}(\alpha_1)|\tau_j(\alpha_1)|^{1/2}, \dots, c\,\boldsymbol{v}^{(r)}(\alpha_d)|\tau_j(\alpha_d)|^{1/2}\Bigr)^T$$

for $1 \le r \le d$, and we replace $c\,\boldsymbol{v}^{(r)}(\alpha_s)|\tau_j(\alpha_s)|^{1/2}$ by $c\sqrt{2}\,\mathrm{Re}(\boldsymbol{v}^{(r)}(\alpha_s))|\tau_j(\alpha_s)|^{1/2}$ and

$$c\boldsymbol{v}^{(r)}(\alpha_t)|\tau_j(\alpha_t)|^{1/2} \quad \text{by} \quad c\sqrt{2}\,\mathrm{Im}(\boldsymbol{v}^{(r)}(\alpha_s))|\tau_j(\alpha_s)|^{1/2}$$

in the construction of $\boldsymbol{T}\boldsymbol{v}^{(r)}$ if $\alpha_s = \bar{\alpha}_t$. The constant c should be calculated from

$$c^d = \frac{L_{\max}}{N(\mathcal{I})\ (\mathrm{Disc}(f/c_d))^{1/2}\ (N(\tau_j))^{1/2}}.$$

In the next iteration step we avoid factoring $\mathcal{Q}$ into prime ideals by including this ideal as a factor of the new $\mathcal{I}$. Its basis in HNF is found by using (9.7). Furthermore we need the embeddings of τ_j for the second LLL reduction. Using (9.2) we can calculate $|\tau_{j+1}(\alpha_l)|$ from $|\tau_j(\alpha_l)|$ and $|\eta_j(\alpha_l)|^2$. We stop with the iterative process when the norms of numerator and denominator and the embeddings of τ_{j+1} are small enough.

Next we calculate the square root $\sqrt{\tau_{j+1}}$ with help of PARI. We first write τ_{j+1} as a polynomial in α. We construct an integer t, being the product of the index and the norms of all ideals which are still in the denominator of τ_{j+1}. Hence $t\tau_{j+1}$ is a polynomial of degree d in α with coefficients in $\mathbb{Z}$. During the algorithm we keep track of the coefficients of the numerator and the denominator of τ_j as a polynomial in α of degree $< d$ modulo several large primes. We use this to express the final $t\tau_{j+1}$ as a polynomial in α of degree $< d$ modulo these primes and we use the Chinese Remainder Theorem to find its coefficients in $\mathbb{Z}$. We divide this element of $\mathbb{Z}[\alpha]$ by t and find its square root by using the method mentioned in [Cohen 1993, § 3.6.2]. This completes the computation of ν.

We now apply the homomorphism φ_i, for $i = 1, 2$, to the expressions found for ν_i:

$$\varphi_i(\nu_i) = \sqrt{\tau_{j+1}}(m)\frac{\prod_{l=1}^{\lfloor\frac{j+1}{2}\rfloor}\eta_{2l-1}(m)}{\prod_{l=1}^{\lfloor\frac{j}{2}\rfloor}\eta_{2l}(m)} \bmod n.$$

Here the $\eta_i(m) \bmod n$ are calculated and multiplied with the $\eta_j(m) \bmod n$, for $j < i$, straight after η_i has been calculated, so there is no need to store a history. We calculate $\gcd(\varphi_1(\nu_1) - \varphi_2(\nu_2), n)$ and hope to find a nontrivial factor of n.

In practice the second lattice basis reduction applied to the $\boldsymbol{T}\boldsymbol{v}^{(r)}$ will yield linear combinations of the $\boldsymbol{v}^{(1)}(\alpha_i), \dots, \boldsymbol{v}^{(d_i)}(\alpha_i)$ with small coefficients. Therefore we can round the entries of the $\boldsymbol{T}\boldsymbol{v}^{(r)}$ without introducing a lot of round-off accumulation. This is the reason why we do not perform one single lattice basis reduction. Now both reductions use integer arithmetic.

It is important for the speed of the algorithm to select the sets $\mathcal{S}_1, \mathcal{S}_2, \mathcal{T}_1$ and $\mathcal{T}_2$ such that we get as much cancellation of primes $p_{(r_1:r_2)}$ as possible. We start with putting half the number of relations of $\mathcal{S}$ and half the number of free primes of $\mathcal{T}$ in $\mathcal{S}_1$ and $\mathcal{T}_1$, respectively, and the rest in $\mathcal{S}_2$ and $\mathcal{T}_2$. While we read in all relations and free primes for one of the polynomials, $f_i(x)$ say, and accumulate the prime ideal factorization of the numerator and the denominator of ν_i, we decide whether it is profitable to put the current relation (a, b) or free prime p in the denominator while it was originally scheduled for the numerator (or vice-versa). If we decide to do so, then we put this relation for this f_i in the denominator and compensate this by multiplying the final $\varphi_i(\nu_i)$ with $a - bm \bmod n$ or $p \bmod n$ respectively.

When using PARI for calculations in number fields it is necessary to use the function `initalg`, which calculates amongst others an integral basis. This function needs to factor the discriminant of the polynomial, which can be too hard for PARI. We solve this problem by factoring the discriminant ourselves and giving the primes to PARI with the function `addprimes`.

10. EXPERIMENTAL RESULTS

In this section we summarize factorization runs for several integers of up to 162 digits, indicating the time spent on each major step of the algorithm.

Except for $72^{99}+1$, all numbers were initially exposed to trial division and the elliptic curve method [Cohen 1993, § 10.3] to find the factors below 40 digits. Thus, in the tables, "C98 from $7^{128}+6^{128}$" means a 98-digit divisor of $7^{128}+6^{128}$ obtained by elimination of "small" primes (3329 and 7329793). If we found only a few small factors we applied the SNFS; otherwise we applied the GNFS.

The numbers factored with the SNFS are listed below, and the relevant statistics appear in Table 1. Figure 1 on page 236 showed the dependence on a and b of the number of relations found, for the C119 from $3^{319}-1$. Figure 2 shows the dependence on b of the number of relations, and Figure 3 the average computation time per relation, for the C98 from $7^{128}+6^{128}$.

The numbers factored with the GNFS are listed below, and the relevant statistics appear in Table 2. Figure 4 shows, for the C97 from $12^{441}+1$, the effect that varying the printing bounds $W_1 = W_2$ has on the time needed for the square root step. The square root program needs the full factorization of both integers $F_i(a,b)$ of the relations in $\mathcal{S}$. Primes larger than W_i are printed in the input file, while smaller ones have to be found with trial division. With large printing bounds trial divisions become very time-consuming. This dependence explains the varying results for square root timings in the preceding tables.

C98 from $7^{128}+6^{128}$, factored into two primes of 49 and 50 digits,
10660 05182 57236 29640 65225 03966 77363 03849 50429 0049 and
57198 45548 36062 92671 60809 76987 21205 78590 71581 43489.

C106 from $2^{543}-1$, factored into two primes of 42 and 64 digits,
53495 53853 19592 51122 74191 75872 57602 50633 51 and
23078 80312 51405 03174 34773 23375 33794 87634 08223 08108 08744 50183 6223.

C119 from $3^{319}-1$, factored into two primes of 41 and 79 digits,
26425 38742 14904 71188 79373 47631 77943 61332 9 and
27428 52582 18630 29446 25818 32587 63622 21386 27169 38311 70052 36010 98185 10078 84358 4437.
Note that $g_1(x) = x^{10}+x^9+\cdots+x+1$, $g_2(x) = x-3^{29}$, and $m = 3^{29}$ satisfy the requirements. Now express $g_1(x)/x^5$ as a polynomial in $x+(1/x)$.

C123 from $2^{511}-1$, factored into two primes of 57 and 67 digits,
14478 09741 87086 26090 39350 34761 41374 56436 36578 29092 41504 17 and
25375 99745 02551 91341 56761 16426 75919 13521 83553 55292 24725 59253 86581 53.
Note that $g_1(x) = x^6+x^5+\cdots+x+1$, $g_2(x) = x-2^{73}$, and $m_g = 2^{73}$ satisfy the requirements. Expressing $g_1(x)/x^3$ as a polynomial in $x+(1/x)$ yields $h_1(x) = x^3+x^2-2x-1$ with $m_h = 2^{73}+2^{-73}$. Use $m_h = 2((2^{36}+2^{-37})^2-1)$ to rewrite $h_1(x)$ as a polynomial in $2^{36}+2^{-37}$. This yields $k_1(x) = 8x^6-20x^4+12x^2-1$ and $m_k = 2^{36}+2^{-37}$. Finally $f_1(x) = 8k_1(x/2)$, so $m_f = 2^{37}+2^{-36}$.

C135 from $73^{73}+1$, factored into two primes of 55 and 80 digits,
45963 69165 58529 11123 52829 63785 23391 57090 14470 88078 32677 and
30968 64234 93721 68556 02856 08720 92954 38228 95151 06526 40624 34659 21744 90064 58993 26733.

The **C162** $(12^{151}-1)/11$, factored into two primes of 44 and 119 digits,
16537 23785 15646 88924 26140 70416 48853 99065 7743 and 49717 86780 03233 78818 76339 90059 60016\
48747 65983 49539 21156 97470 05759 15322 82419 11167 04320 09270 16884 28573 10302 48831 34912 6419.

Numbers factored with the SNFS.

	C98	C106	C119
factor of	$7^{128}+6^{128}$	$2^{543}-1$	$3^{319}-1$
$f_1(x)$	x^4+1	$4x^4+2x^2+1$	$x^5+x^4-4x^3-3x^2+3x+1$
$f_2(x)$	$6^{32}x-7^{32}$	$x-2^{90}$	$3^{29}x-3^{58}-1$
m	$7^{32}6^{-32} \bmod n$	2^{90}	$3^{29}+3^{-29} \bmod n$
sieving region	$\|a\| \le 2\cdot 10^6$	$\|a\| \le 3.5\cdot 10^5$	$\|a\| \le 16\cdot 10^5$, $1 \le b \le 103000$;
	$1 \le b \le 16\cdot 10^3$	$1 \le b \le 10^5$	$\|a\| \le 12\cdot 10^5$, $103001 \le b \le 345000$
B_1	$1.6\cdot 10^6$	$5\cdot 10^5$	10^6
B_2	$1.6\cdot 10^6$	$8.1\cdot 10^5$	$1.4\cdot 10^6$
L_1	$3\cdot 10^7$	$12\cdot 10^6$	$2\cdot 10^7$
L_2	$3\cdot 10^7$	$12\cdot 10^6$	$2.5\cdot 10^7$
sieving time	450 hours	250 hours	800 hours
# sieving rel.	2,337,618	1,106,949	2,221,686
# filter rel. / sets	982,672/587,076	264,583/126,254	774,265/349,961
matrix size	$539,020\times 620,650$	$128,546\times 133,738$	$348,852\times 367,182$
bl. Lanczos time	74 min	6 min	38 min
square root time	69 min	47 min	62 min
# trials	1	2	1

	C123	C135	C162
factor of	$2^{511}-1$	$73^{73}+1$	$12^{151}-1$
$f_1(x)$	$x^6-10x^4+24x^2-8$	x^5+73^2	$12x^5-1$
$f_2(x)$	$2^{36}x-2^{73}-1$	$x-73^{15}$	$x-12^{30}$
m	$2^{37}+2^{-36} \bmod n$	73^{15}	12^{30}
sieving region	$\|a\| \le 6\cdot 10^5$	$\|a\| \le 2\cdot 10^6$	?
	$1 \le b \le 37\cdot 10^4$	$1 \le b \le 2.6\cdot 10^5$	
B_1	$15\cdot 10^5$	$2\cdot 10^6$	?
B_2	$11\cdot 10^5$	$2\cdot 10^6$	?
L_1	$3\cdot 10^7$	$3\cdot 10^7$	10^8
L_2	$2.5\cdot 10^7$	$3\cdot 10^7$	10^8
sieving time	700 hours	2150 hours	see caption
# sieving rel.	1,901,187	2,746,848	$8.98\cdot 10^6$
# filter rel. / sets	420,896/222,014	1,154,111/583,631	1,807,808/822,361
matrix size	$430,018\times 439,058$	$581,870\times 590,573$	$828,077\times 833,017$
bl. Lanczos time	97 hours	92 min	205 min
square root time	13 hours	130 min	10.5 hours
# trials	1	1	2

TABLE 1. Statistics of SNFS runs. The square root timings are given for one dependency. "Sieving time" is a rough estimate on an SGI Indy workstation (100 MHz R4000SC processor), except for the C162, where it represents 8 weeks of idle time on 30 workstations at Oregon State University, Corwallis (USA). "Block Lanczos time" is time on one processor of a Cray C98, except for the C123, where it is on one processor of an SGI Challenge (150 MHz R4400SC processor). "Square root time" is always on one 150 MHz R4400SC processor of an SGI Challenge. A question mark indicates that records have been lost. See also Figure 1 for the C119 and Figures 2 and 3 for the C98.

ACKNOWLEDGEMENTS

I am particularly grateful to P. L. Montgomery for helping me understand the Number Field Sieve and for sharing with me his NFS program, developed partially by A. K. Lenstra and Oregon State University. I thank H. W. Lenstra, Jr., P. L. Montgomery, H. J. J. te Riele, and R. Tijdeman for reading the paper and for suggesting several improvements. I also thank the referee for remarks that helped improve the presentation in Section 10.

This work was sponsored by the Stichting Nationale Computerfaciliteiten (National Computing Facilities Foundation, NCF) for the use of supercomputer facilities, with financial support from the Nederlandse Organisatie voor Wetenschappelijk Onderzoek (Netherlands Organization for Scientific Research, NWO).

C87 from $72^{99}+1$, factored into two primes of 28 and 59 digits,
80975 40789 16899 09106 86588 841 and 16798 25963 43052 65460 77643 18240 65479 28992 59252 26507 26238 4081.

15112 89658 85928 67299 7 x^2 − 14454 82064 03710 35730 30 x − 82891 10711 41503 38627 28
10193 02053 49942 54454 47 x^2 + 69591 00565 51957 53864 28 x − 73926 15678 48940 75263 915
12971 49181 97797 46345 57652 15830 55986 93259 36651 20309 00057 23712 50151 97604 40714 27556 35889 06

72987 9993 x^2 + 19574 91701 87826 93290 2 x − 77557 88285 64247 72330 65859 26647 4816
21732 17947 x^2 − 79076 05467 51140 42573 56 x − 23092 86246 66134 62049 23942 78524 71175
91584 63560 41789 38356 37052 93002 21754 09773 28747 97358 88020 12186 39497 14431 09944 13684 20046 6

C97 from $12^{441}-1$, factored into two primes of 44 and 53 digits,
80563 59586 39915 46010 66111 58818 06884 96700 4027 and
34988 88222 91413 72324 91945 65798 20527 95623 42753 61821 641.

−30177 43825 3 x^2 + 43549 18245 10787 39544 8959 x + 35629 87893 56423 22360 48460 61872 83613 50
−53619 92778 05 x^2 − 48039 40130 45794 97453 61008 x + 63307 94281 94378 06775 38097 94819 80102 858
28793 29914 27719 83147 13924 73244 89082 52443 26896 13883 6873469449 40161 32035 13827 86658 32257 58151 28653 4

C105 from $3^{367}-1$, factored into two primes of 52 and 54 digits,
15114 95257 84007 07169 98865 69402 29379 35039 92823 13504 93 and
50195 39973 89244 52840 42479 06279 09065 41054 68962 12425 1929.

34291 05277 37 x^2 + 86817 06933 35194 65483 64161 2 x + 54075 90626 04782 97135 71395 36186 42487 4771
12420 60255 079 x^2 − 91304 92731 81768 81696 25532 18 x + 12912 87673 00065 23363 11682 29536 26798 24208 00
22914 35905 55869 46906 21150 13538 55768 19231 64235 75426 62177 65793 56350 02756 74926 89398 72232 45481 40116 05440 05942

C106 from $12^{157}+1$ was factored into two primes of 43 and 63 digits,
59076 48479 16668 26924 93875 79080 94200 49637 197 and
78003 47545 94869 44147 52419 18877 75473 24150 95147 75893 20240 91338 271.

19003 04761 13 x^2 − 10164 31637 34436 73606 69602 94 x − 32430 28756 04959 76143 91031 71598 23376 25514 4
−78508 32606 39 x^2 − 40196 86460 51742 27034 42801 72 x + 16408 60800 01456 03417 92387 66543 25668 77138 27
17900 44128 75726 25768 48153 41213 37659 37899 09788 88143 77815 81676 91054 76827 69666 52099 45565 82560 64297 87588 58169 9

C107 from $6^{223}+1$, factored into 2 primes of 48 and 59 digits,
83579 06552 59197 87058 63199 55878 76457 41368 53697 743 and
16660 35981 83855 55664 06860 19793 47465 06370 91184 73978 85209 6987.

−54016 17762 83 x^2 − 42570 42830 28714 25377 92693 15 x − 46786 96410 85791 79806 10186 34789 10720 07155 8
−24179 95148 05 x^2 + 76311 97031 66287 85485 31988 89 x − 31165 39943 59418 67031 97753 30136 43451 35069 86
12637 53059 94677 76761 85312 84126 24277 13734 77298 51839 92404 83922 87605 24925 32707 97264 40981 32306 53725 40515 54848 92

Numbers factored with the GNFS. After each factorization we give the polynomials f_1 and f_2 used in the run, and the integer m. The C87 was factored twice, once with classical sieving (first set of values of f_1, f_2, m) and once with lattice sieving. See also Figure 4 for the C97.

	C87	C87	C97
factor of	$72^{99}+1$	$72^{99}+1$	$12^{441}+1$
sieving method	classical	lattice	lattice
sieving region	$\lvert a\rvert \le 2\cdot 10^6$, $1\le b\le 48\cdot 10^4$	$\lvert a\rvert \le 7.5\cdot 10^{12}$, $b=1$	$\lvert a\rvert \le 25\cdot 10^{12}$, $b=1$
$B_1 = B_2$	$1.6\cdot 10^6$	10^6	$2.2\cdot 10^6$
$L^{(l)}$	$(L_1 = 4\cdot 10^7)$	10^6	$10\cdot 10^6$
$L^{(u)}$	$(L_2 = 4\cdot 10^7)$	$2.346\cdot 10^6$	$L^{(u)} = 24\cdot 10^6$
sieving time	2100 hours	1500 hours	3500 hours
# sieving rel.	3,480,325	521,901	3,599,014
# filter rel. / sets	741,930/338,580	426,241/409,699	1,247,094/604,205
matrix size	$364,215\times 366,907$	$273,475\times 437,441$	$637,711\times 644,950$
bl. Lanczos time	41 min$^{(2)}$	26 min$^{(2)}$	123 min$^{(2)}$
square root time	128 min$^{(3)}$	36 min$^{(3)}$	78 min$^{(3)}$
# trials	1	2	2

	C105	C106	C107
factor of	$3^{367}-1$	$12^{157}+1$	$6^{223}+1$
sieving method	lattice	lattice	lattice
sieving region	$\lvert a\rvert \le 7.5\cdot 10^{14}$, $b=1$	$\lvert a\rvert \le 10^{15}$, $b=1$	$\lvert a\rvert \le 10^{15}$, $b=1$
$B_1 = B_2$	$1.6\cdot 10^6$	$2.7\cdot 10^6$	$2.9\cdot 10^6$
$L^{(l)}$	$23\cdot 10^6$	$2.7\cdot 10^7$	$2.72\cdot 10^7$
$L^{(u)}$	$30\cdot 10^6$	$3\cdot 10^7$	$3\cdot 10^7$
sieving time	see caption	11900 hours	11200 hours
# sieving rel.	$3.59\cdot 10^6$	3,272,224	3,098,987
# filter rel. / sets	?/1,218,633	2,151,431/1,191,636	2,152,685/1,155,270
matrix size	$1,284,719\times 1,294,861$	$1,266,098\times 1,295,043$	$1,226,577\times 1,252,846$
bl. Lanczos time	439 min$^{(2)}$	423 min$^{(2)}$	421 min$^{(2)}$
square root time	4.8 hours$^{(3)}$	2.0 hours$^{(4)}$	2.1 hours$^{(3)}$
# trials	1	1	5

TABLE 2. Statistics of GNFS runs. The square root timings are given for one dependency. "Sieving time" is a rough estimate on an SGI Indy workstation (100 MHz R4000SC processor), except for the C105, where it represents 8 weeks of idle time on 40 workstations at Oregon State University, Corwallis (USA). "Block Lanczos time" is time on one processor of a Cray C98. "Square root time" is on one processor of an SGI Challenge (150 MHz R4400SC processor), except for the C106, where the clock rate was 200 MHz. A question mark indicates that records have been lost. See also Figure 4 for the C97.

REFERENCES

[Adleman 1991] L. M. Adleman, "Factoring numbers using singular integers", pp. 64–71 in *Proceedings 23rd Annual ACM Symposium on Theory of Computing* (New Orleans, 1991), ACM, New York, 1991.

[Batut et al. 1995] C. Batut, D. Bernardi, H. Cohen and M. Olivier, *User's Guide to Pari-GP*. This manual is part of the program distribution, available by anonymous ftp from the host megrez.math.u-bordeaux.fr.

[Buhler et al. 1993] J. P. Buhler, H. W. Lenstra, Jr., and C. Pomerance, "Factoring integers with the number field sieve", pp. 50–94 in [Lenstra et al. 1993a].

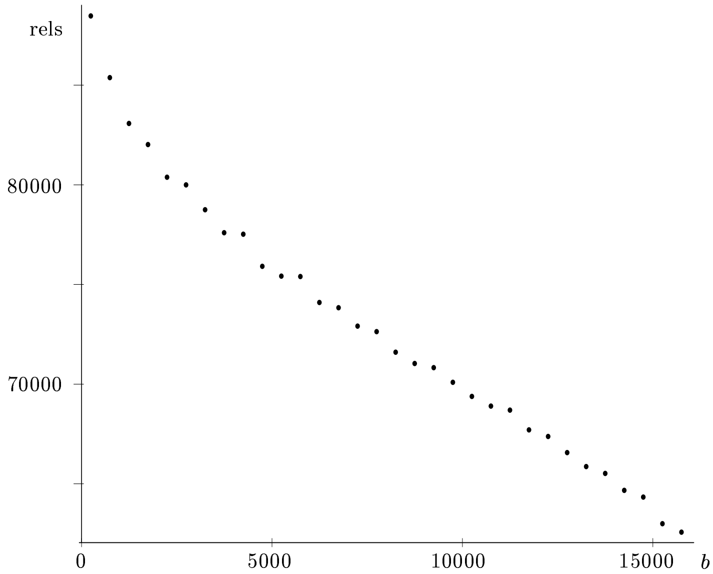

FIGURE 2. Number of relations found in each range of 500 b-values for the factorization of the C98 from $7^{128} + 6^{128}$.

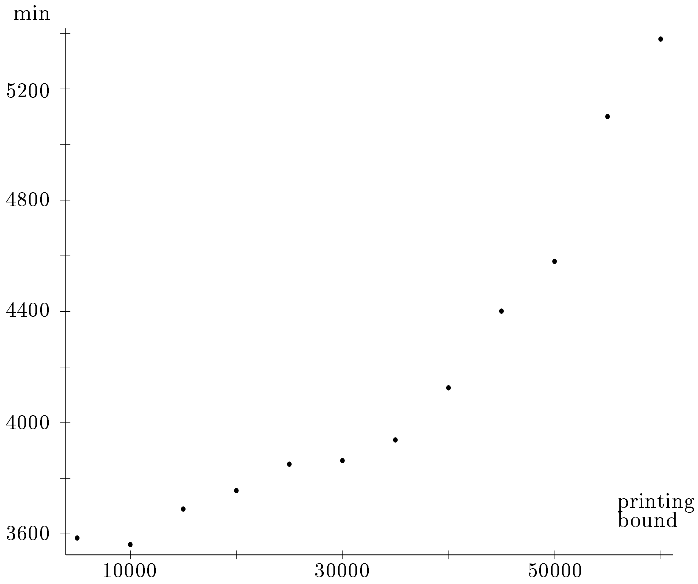

FIGURE 4. Square root timings (on one 150 MHz R4400SC processor of an SGI Challenge) for the C97 from $12^{441} + 1$.

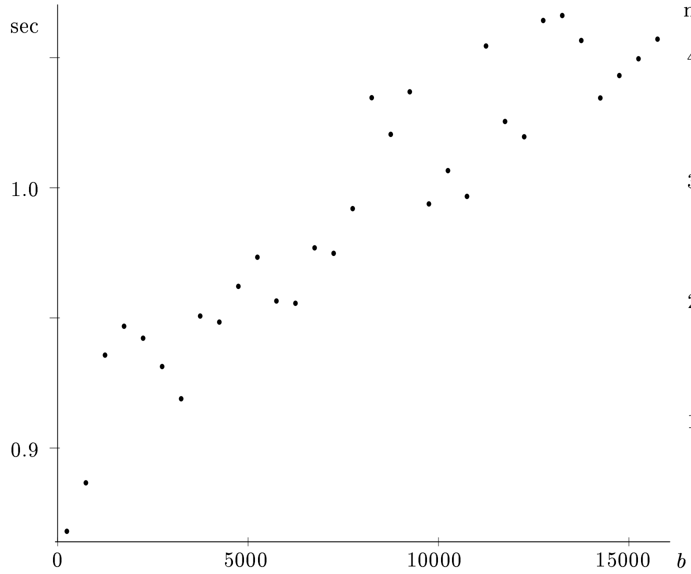

FIGURE 3. Approximate average time per relation needed in the factorization run of the C98 from $7^{128} + 6^{128}$, again as a function of b. Times are on an SGI Indy workstation with a 100 MHz R4000SC processor.

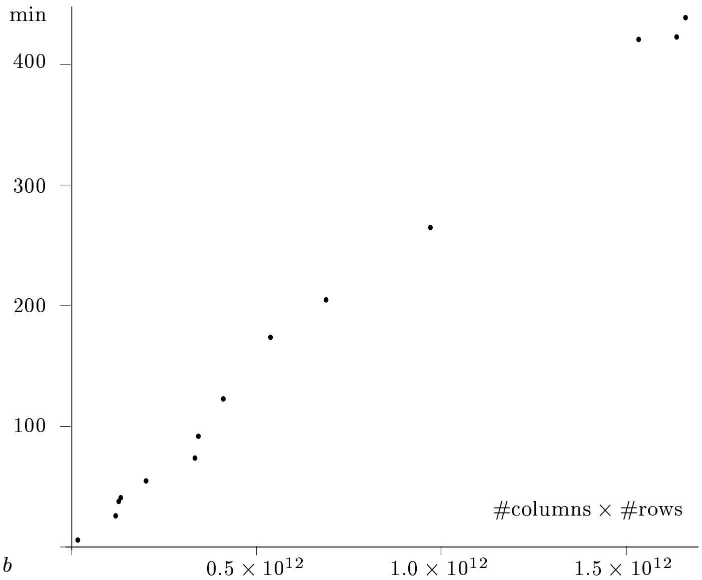

FIGURE 5. All matrices from the experiments had close to the average of 30.3 nonzero elements per row. Here we show the dependence of the block Lanczos timings on the product of the number of rows and columns.

[Buhler et al. 1994] J. P. Buhler, P. L. Montgomery, R. Robson, and R. Ruby, "Technical report implementing the number field sieve", Oregon State University, Corwallis, OR, 1994.

[Cohen 1993] H. Cohen, *A Course in Computational Algebraic Number Theory*, Springer, Berlin, 1993.

[Coppersmith 1993] D. Coppersmith, "Solving linear equations over GF(2): Block Lanczos algorithm", *Linear Algebra and its Applications* **192** (1993), 33–60.

[Couveignes 1993] J.-M. Couveignes, "Computing a square root for the number field sieve", pp. 95–102 in [Lenstra et al. 1993a].

[Frobenius 1896] F. G. Frobenius. "Über Beziehungen zwischen den Primidealen eines algebraischen Körpers und den Substitutionen seiner Gruppe", *Sitzungsberichte der Königlich Preußischen Akademie der Wissenschaften zu Berlin* (1896), pp. 689–703. Reprinted in *Gesammelte Abhandlungen*, Band II, Springer, Berlin, 1968.

[Golliver et al. 1994] R. A. Golliver, A. K. Lenstra, and K. S. McCurley, "Lattice sieving and trial division", pp. 18–27 in *Algorithmic Number Theory* (edited by L. M. Adleman and M.-D. Huang), Lecture Notes in Comp. Sci. **877**, Springer, Berlin, 1994.

[Knuth 1981] D. E. Knuth, *The Art of Computer Programming*, vol. 2: *Seminumerical Algorithms*, 2nd ed., Addison–Wesley, Reading, MA, 1981 (or 3rd ed., 1997).

[LaMacchia et al. 1991] B. A. LaMacchia and A. M. Odlyzko, "Solving large sparse linear systems over finite fields", pp. 109–133 *Advances in Cryptology: CRYPTO '90* (edited by A. J. Menezes and S. A. Vanstone), Lecture Notes in Comp. Sci. **537**, Springer, Berlin, 1991.

[Lang 1970] S. Lang, *Algebraic Number Theory*, Addison–Wesley, Reading, MA, 1970.

[Lenstra et al. 1993a] A. K. Lenstra and H. W. Lenstra, Jr., *The Development of the Number Field Sieve*, Lecture Notes in Math. **1554**, Springer, Berlin, 1993.

[Lenstra et al. 1982] A. K. Lenstra, H. W. Lenstra, Jr., and L. Lovász, "Factoring polynomials with rational coefficients", *Mathematische Annalen*, **261** (1982), 515–534.

[Lenstra et al. 1993b] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard, "The number field sieve", pp. 11–42 in [Lenstra et al. 1993a].

[Lenstra et al. 1993c] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard, "The factorization of the ninth Fermat number", *Mathematics of Computation*, **61** (1993), 319–349.

[Montgomery 1994] Peter L. Montgomery, "Square roots of products of algebraic numbers", pp. 567–571 in *Mathematics of Computation* 1943–1993: *A Half-Century of Computational Mathematics* (edited by Walter Gautschi), Proceedings of Symposia in Applied Mathematics, American Math. Soc., Providence, 1994. Long version to appear.

[Montgomery 1995] Peter L. Montgomery, "A block Lanczos algorithm for finding dependencies over GF(2)", pp. 106–120 in *Advances in Cryptology: Eurocrypt* '95 (edited by L. C. Guillou and J.-J. Quisquater), Lecture Notes in Comp. Sci. **921**, Springer, Berlin, 1995.

[Neukirch 1992] J. Neukirch, *Algebraische Zahlentheorie*, Springer, Berlin, 1992.

[Pollard 1993a] J. M. Pollard, "Factoring with cubic integers", pp. 4–10 in [Lenstra et al. 1993a].

[Pollard 1993b] J. M. Pollard, "The lattice sieve", pp. 43–49 in [Lenstra et al. 1993a].

[Pomerance 1985] C. Pomerance, "The quadratic sieve factoring algorithm", pp. 169–182 in *Advances in Cryptology: Eurocrypt* '84 (edited by T. Beth, N. Cot, and I. Ingemarsson), Lecture Notes in Comp. Sci. **209**, Springer, New York, 1985.

[Riesel 1985] H. Riesel, *Prime Numbers and Computer Methods for Factorization*, Birkhäuser, Boston, 1985.

[Standish 1980] T. A. Standish, *Data Structure Techniques*, Addison–Wesley, Reading, MA, 1980.

Marije Elkenbracht-Huizing, CWI, P.O. Box 94079, 1090 GB Amsterdam, The Netherlands (marije@cwi.nl)

Received August 24, 1995; accepted in revised form April 9, 1996