THE NUMBER FIELD SIEVE

A. K. LENSTRA, H. W. LENSTRA, JR., M. S. MANASSE, J. M. POLLARD

ABSTRACT. The number field sieve is an algorithm to factor integers of the form $r^e - s$ for small positive r and $|s|$. The algorithm depends on arithmetic in an algebraic number field. We describe the algorithm, discuss several aspects of its implementation, and present some of the factorizations obtained. A heuristic run time analysis indicates that the number field sieve is asymptotically substantially faster than any other known factoring method, for the integers that it applies to. The number field sieve can be modified to handle arbitrary integers. This variant is slower, but asymptotically it is still expected to beat all older factoring methods.

1. INTRODUCTION

In this paper we present a novel algorithm to factor integers of the form $r^e - s$, where r and $|s|$ are small positive integers, $r > 1$, and e is large. The algorithm has become known as the *number field sieve*, because it depends on arithmetic in an algebraic number field combined with more traditional sieving techniques. It has proved to be quite practical, its most notable success being the factorization of the ninth Fermat number. We refer to our account [26] of the latter factorization for an introduction to the number field sieve.

Let N be an integer of the form $r^e - s$ as above. It should be thought of as an integer that we want to factor into prime factors. Examples of such N can be found in the Cunningham tables [3]. In many cases, one already knows some prime factors of N, so that it is the cofactor n that remains to be factored. Applying the number field sieve for this purpose is not recommended if n is much smaller than N, since the conjectured run time of the algorithm depends on the size of N rather than on the size of n.

To express the conjectured run time, we define

$$L_x[v, \lambda] = \exp(\lambda(\log x)^v(\log\log x)^{1-v})$$

for real numbers x, v and λ with $x > \mathrm{e}$. In the discussion below we will, for simplicity, abbreviate the expression $L_x[v, \lambda + o(1)]$ to $L_x[v, \lambda]$; here the $o(1)$ is for $x \to \infty$. With this notation, we expect that for r and $|s|$ below a fixed upper bound the number field sieve takes time $L_N[\frac{1}{3}, c]$, where $c = (32/9)^{1/3} \doteq 1.5263$, irrespectively of the size of the factors of N. We are not able to prove this run time rigorously, and even our heuristic argument has a weak spot (see 6.4).

1991 *Mathematics Subject Classification.* Primary 11Y05, 11Y40.

Key words and phrases. Factoring algorithm, algebraic number fields.

The second author was supported by NSF under Grant No. DMS-9002939 and by NSA/MSP under Grant No. MDA90-H-4043.

Buhler and Pomerance observed that the idea of the number field sieve can be applied to general integers as well, i.e., to integers n that do not necessarily have a small multiple of the form $r^e - s$ as above. The generalized version of the number field sieve is also conjectured to take time $L_n[\frac{1}{3}, c]$, but with a larger value for c than above. Due to additional contributions by Adleman and Coppersmith, the smallest value for c that can currently conjecturally be achieved is given by

$$c = \frac{(92 + 26\sqrt{13})^{1/3}}{3} \doteq 1.9019,$$

see Section 9, and [1; 7; 11]. This makes the number field sieve, conjecturally and asymptotically, into the fastest currently known integer factoring algorithm; only the elliptic curve method [29] is, for a special class of numbers, supposed to be faster.

The function $L_n[v, \lambda]$, which plays an important role in the analysis of modern factoring algorithms, interpolates between powers of n and powers of $\log n$. More specifically, we have

$$L_n[1, \lambda] = n^{\lambda}, \qquad L_n[0, \lambda] = (\log n)^{\lambda},$$
$$\log\log L_n[v, \lambda] = v \cdot \log\log L[1, \lambda] + (1 - v) \cdot \log\log L[0, \lambda].$$

The most significant parameter is v. Traditional algorithms such as trial division have $v = 1$, in the sense that they run in time $L[1, \lambda]$ for some $\lambda > 0$. These algorithms are said to run in exponential time. A polynomial time algorithm would have $v = 0$. Thus, the many algorithms that run in time $L_n[\frac{1}{2}, \lambda]$ (see below) are in this sense halfway between the exponential time and polynomial time ones. The number field sieve, which has $v = \frac{1}{3}$, represents an additional step in the direction of polynomial time algorithms. The notation $L_n[v, \lambda]$ was introduced in [25], following the notation $L(n)^{\lambda}$ that Pomerance introduced for $L_n[\frac{1}{2}, \lambda]$ in 1983 (see [34]).

The first person to realize that the function $L_n[\frac{1}{2}, \lambda]$ can be used to express conjectural run times of factoring algorithms was Schroeppel, in 1975 (see [21, Section 4.5.4]). In 1978 Dixon obtained a rigorous result of this nature (see [16]). The study of the precise value of λ was initiated by Pomerance (see [34]). It is now conjectured that many factoring algorithms, including the continued fraction method, the quadratic sieve, and the elliptic curve method, run in expected time at most $L_n[\frac{1}{2}, 1]$, and for the class group relations method this has been proved (see [31]). Accounts of these developments can be found in [25] and [36]. The *cubic sieve* algorithm (see [14, Section 7; 25, Section 4.E]) is conjectured to be faster and to run in time $L_n[\frac{1}{2}, c']$, with $\sqrt{2/3} \leq c' < 1$; however, it applies only to numbers of a special form, including Cunningham numbers, and it has never proved to be of more than theoretical interest.

In 1981 the function $L_n[\frac{1}{3}, \lambda]$ made its appearance in the analysis of factoring algorithms, when Schnorr [38] showed, under plausible assumptions, that an integer n can be factored in time $L_n[\frac{1}{3}, 2]$ provided that suitable lists of *smooth numbers* are available. Here we call a number *smooth*—the term is due

to Rivest—if its prime factors are small; more precisely, it is *B-smooth* if its prime factors are at most B.

The importance of the function $L_n[v, \lambda]$ for analyzing factoring algorithms is due to its connection with smooth numbers. Many factoring algorithms proceed by generating a sequence of integers, in a more or less random fashion, of which only the smooth ones are useful. For all algorithms before the number field sieve, the integers that are inspected for smoothness have order of magnitude n^w, for some constant w that depends on the algorithm. For instance, all algorithms with expected run time $L_n[\frac{1}{2}, 1]$ have $w = \frac{1}{2}$. The cubic sieve algorithm was the first to break through the $w = \frac{1}{2}$ barrier with $w = \frac{1}{3}$ for n of a special form, and thus achieved run time $L_n[\frac{1}{2}, c']$ with $c' < 1$ for such n. A more dramatic improvement is realized by the number field sieve: the integers that it inspects for smoothness are only $n^{o(1)}$, if n is not too much smaller than N. This makes it the first factoring algorithm with conjectured run time essentially faster than $L_n[\frac{1}{2}, \lambda]$, for any positive constant λ. We refer to [7, Section 10] for a discussion of the relation between the run time of a factoring algorithm and the size of the numbers that it inspects for smoothness, and for an explanation of the role played by the function $L_n[v, \lambda]$.

From an algorithmic point of view, the problem of factoring integers is closely related to the discrete logarithm problem, see [25; 36]. The conjectural run time of many discrete logarithm algorithms for a finite field of n elements is of the form $L_n[v, \lambda]$ with $v = \frac{1}{2}$. Coppersmith was the first to achieve $v = \frac{1}{3}$, for the case that n is a power of 2 (see [10; 25, Section 3.17; 19]). His *bimodal polynomials* method shares a few formal features with Schnorr's work [38]: the appearance of $v = \frac{1}{3}$, and the requirement that two expressions are simultaneously smooth. It is interesting to observe that the number field sieve has these features as well. In addition, both the number field sieve and the bimodal polynomials method start by looking for a good auxiliary polynomial.

The algorithm in the present paper was inspired by the discrete logarithm algorithm for prime n using Gaussian integers that was presented by Coppersmith, Odlyzko, and Schroeppel (see [14, Section 7; 23]). This algorithm, which has $v = \frac{1}{2}$, was in turn inspired by work of ElGamal [17]. The main change that we made, which is crucial for obtaining $v = \frac{1}{3}$, is that we use rings of algebraic integers in higher degree number fields, and that we optimize the choice of the degree as a function of the number to be factored (see 6.3). Gordon [18] showed that the same technique can be used for the discrete logarithm problem for prime n. The conjectural run time estimate of his algorithm is $L_n[\frac{1}{3}, 9^{1/3}]$, where $9^{1/3} \doteq 2.0801$. Schirokauer [39] improved this to $L_n[\frac{1}{3}, (64/9)^{1/3}]$, where $(64/9)^{1/3} \doteq 1.9230$.

In Section 2 of the present paper we describe the number field sieve, as it applies to integers of the form $r^e - s$ for small positive r and $|s|$. Details that are left out from this description are explained in Sections 3, 4, and 5. A conjectural analysis of the run time of the number field sieve is given in Section 6. In Section 7 we discuss a few modifications to the algorithm. Examples of factorizations that have been obtained by means of the number field sieve are presented in Section 8.

Section 9 is devoted to possible generalizations of the number field sieve to arbitrary integers.

In our description of the number field sieve we will make a few simplifying and not always realistic assumptions about the number fields that we are using, cf. 2.5 and 3.1. We refer to 3.4–3.8, Section 9, and [2; 7] for variations of the number field sieve that make no simplifying assumptions about the fields involved.

We shall denote by $\mathbf{Z}$ the ring of integers, and by $\mathbf{Q}$, $\mathbf{R}$, and $\mathbf{C}$ the fields of rational, real, and complex numbers, respectively.

2. The algorithm

Let n be an odd integer, $n > 1$, and assume that n is not a prime number or a power of a prime number. We assume that n itself or a small multiple of n is of the form $r^e - s$, for a small integer $r > 1$ and a non-zero integer s of small absolute value, and with e an integer that is possibly much larger. It is assumed that r, e, and s are given along with n. Numbers of this form often appear on the 'wanted' lists from [3]. The fact that n is not a prime number can usually easily be proved by means of a probabilistic compositeness test, see [25, Section 5.1]. If we use the variation of the probabilistic compositeness test described in [26, Section 2.5] we can also easily check that n is not a prime power. We describe a factoring algorithm, the *number field sieve*, that makes use of the special form of the multiple $r^e - s$ of n, to factor n. For background on the elementary algebraic number theory used by the algorithm we refer to [40] and to [26, Section 4].

2.1. *Outline of the algorithm.* For a random integer x satisfying

$$x^2 \equiv 1 \bmod n \tag{2.2}$$

there is a probability of at least $\frac{1}{2}$ that $\gcd(n, x-1)$ is a non-trivial factor of n. To factor n it therefore suffices to construct several solutions x to (2.2) in an apparently random manner. Many factoring algorithms, including the number field sieve, achieve this by means of the following three-step approach.

Step 1. *Selecting the factor base.* Select a collection of non-zero elements $a_i \in \mathbf{Z}/n\mathbf{Z}$, with i ranging over some finite index set I. The collection $(a_i)_{i\in I}$ is called the *factor base*, for a reason that will be clear from the sequel. The elements of the factor base should not be confused with a possible list of candidate factors of n; indeed, we may assume that all a_i are units in $\mathbf{Z}/n\mathbf{Z}$, because if they are not, then n can be factored immediately.

Step 2. *Collecting relations.* Collect *relations* between the a_i, i.e., vectors $v = (v_i)_{i\in I} \in \mathbf{Z}^I$ for which

$$\prod_{i\in I} a_i^{v_i} = 1. \tag{2.3}$$

Stop as soon as the collection V of relations that have been found contains slightly more than $\#I$ elements.

Step 3. *Finding dependencies.* Find dependencies modulo 2 among the elements of V, i.e., subsets W of V such that $\sum_{v\in W} v = 2 \cdot (w_i)_{i\in I}$ with $w_i \in \mathbf{Z}$.

Notice that non-trivial dependencies exist because $\#V > \#I$. For each dependency W we can calculate an integer x with $\prod_{i \in I} a_i^{w_i} = (x \bmod n)$, and this integer satisfies (2.2). Under conditions on the a_i and V that can usually not be proved but that are normally satisfied, consideration of a few linearly independent W's leads to the complete factorization of n into powers of distinct prime numbers; see [26, Section 2.6] for a further discussion of this point.

The remainder of this section is devoted to a description of how Steps 1 and 2 are carried out in the number field sieve. For Step 3 we refer to the literature on large sparse matrix elimination, cf. [22; 37; 42; 12; 13], and to [26].

2.4. *The idea of the number field sieve.* The number field sieve is based on the observation that it is possible to construct a number field $K = \mathbf{Q}(\alpha)$ and a ring homomorphism φ from the subring $\mathbf{Z}[\alpha]$ of K to $\mathbf{Z}/n\mathbf{Z}$ such that $\varphi(\alpha) = (m \bmod n)$, where both α and $|m|$ are small compared to n; here the smallness of α is measured by means of the sum of the absolute values of the coefficients of its irreducible polynomial, which are supposed to be integers. The idea is then to look for pairs of small coprime integers a and b such that both the algebraic integer $a + b\alpha$ and the integer $a + bm$ are smooth, in a sense to be specified below. Because $\varphi(a + b\alpha) = (a + bm \bmod n)$, each pair provides an equality of two products in $\mathbf{Z}/n\mathbf{Z}$. The factors occurring in these products form the factor base, and each congruence leads to a relation as in (2.3).

2.5. *Construction of the number field.* To define the number field K and the homomorphism φ as in 2.4 we proceed as follows. Given the multiple $r^e - s$ of n, we first select a small positive integer d that will serve as the extension degree. More about the choice of d can be found in Sections 6, 7, and 8. Given d, let k be the least positive integer for which $k \cdot d \geq e$, put $t = s \cdot r^{k \cdot d - e}$, and let f be the polynomial $X^d - t$. The number $m = r^k$ satisfies $f(m) \equiv 0 \bmod n$, since n divides $r^e - s$. Our number field K is now given by $K = \mathbf{Q}(\alpha)$, where $f(\alpha) = 0$.

We will assume that the polynomial f is irreducible. This condition is likely to be satisfied, since in realistic cases a non-trivial factor of f gives rise to a non-trivial factor of n. If it is not satisfied we can replace f by a suitable factor. The irreducibility of f is easily checked: f is reducible if and only if either there is a prime number p dividing d such that t is a pth power, or 4 divides d and $-4t$ is a fourth power (see [24, Chapter VI, Theorem 9.1]). For example, if r is not a power of a smaller integer, and $s = 1$, then f is irreducible if and only if $\gcd(d, e) = 1$.

The irreducibility of f implies that the degree of the number field K equals d, and that each element of K has a unique expression of the form $\sum_{i=0}^{d-1} q_i \alpha^i$ with $q_i \in \mathbf{Q}$. The subring $\mathbf{Z}[\alpha]$ of K consists of the expressions $\sum_{i=0}^{d-1} s_i \alpha^i$ with coefficients $s_i \in \mathbf{Z}$. The ring homomorphism $\varphi\colon \mathbf{Z}[\alpha] \to \mathbf{Z}/n\mathbf{Z}$ is now defined by $\varphi(\alpha) = (m \bmod n)$. Generally, we have $\varphi(\sum_{i=0}^{d-1} s_i \alpha^i) = (\sum_{i=0}^{d-1} s_i m^i \bmod n)$ if $s_i \in \mathbf{Z}$.

To simplify the exposition of the algorithm, we will assume that *the ring* $\mathbf{Z}[\alpha]$ *is a unique factorization domain.* As we shall see in 3.4, this is a strong assumption, which is not always satisfied. We refer to Section 3 for a discussion

of the modifications that are necessary if $\mathbf{Z}[\alpha]$ is not a unique factorization domain.

To give an example, for $3^{239}-1$ (one of the numbers we factored, cf. Section 8), we used $d = 5$ as extension degree, $m = 3^{48}$, and α a zero of the polynomial $f = X^5 - 3$; in this case $\mathbf{Z}[\alpha]$ is indeed a unique factorization domain. For another number we factored, $2^{512}+1$, we used $d = 5$, $m = 2^{103}$, and α a zero of $X^5 + 8$. Because $\alpha^2/2 \notin \mathbf{Z}[\alpha]$ and because $\alpha^2/2$ is a zero of $X^5 - 2 \in \mathbf{Z}[X]$, we find that $\mathbf{Z}[\alpha]$ is not the ring of integers of K. We simply got around that problem by using $\mathbf{Z}[\alpha^2/2]$ instead of $\mathbf{Z}[\alpha]$ in the algorithms described below.

It is in the construction of K and φ, as described above, that we exploit the special form of the multiple $r^e - s$ of n. The main difficulty with general n is that one is led to consider much "larger" number fields, which are much harder to control. This difficulty is discussed in Section 9.

2.6. *Smoothness in the number field.* An algebraic integer is called *B-smooth* if every prime number dividing its norm is at most B. We shall mainly be interested in smoothness of algebraic integers of the form $a + b\alpha$, where a, b are coprime integers. The norm $\mathbf{N}(a + b\alpha)$ of $a + b\alpha$ is equal to $a^d - t(-b)^d$, so $a + b\alpha$ is B-smooth if and only if $|a^d - t(-b)^d|$ is a product of prime numbers $\leq B$.

The *norm* $\mathfrak{N}\mathfrak{a}$ of a non-zero ideal $\mathfrak{a}$ of $\mathbf{Z}[\alpha]$ is defined by $\mathfrak{N}\mathfrak{a} = \#(\mathbf{Z}[\alpha]/\mathfrak{a})$, which is a positive integer. A *first degree prime ideal* of $\mathbf{Z}[\alpha]$ is a non-zero ideal $\mathfrak{p}$ of prime norm p. For such an ideal we have $\mathbf{Z}[\alpha]/\mathfrak{p} \cong \mathbf{Z}/p\mathbf{Z}$, which is a field, so that $\mathfrak{p}$ is indeed a prime ideal. The set of first degree prime ideals $\mathfrak{p}$ is in bijective correspondence with the set of pairs $(p, c \bmod p)$, where p is a prime number and $c \in \mathbf{Z}$ satisfies $f(c) \equiv 0 \bmod p$; if $\mathfrak{p}$ corresponds to $(p, c \bmod p)$, then $\mathfrak{N}\mathfrak{p} = p$, the map $\mathbf{Z}[\alpha] \to \mathbf{Z}[\alpha]/\mathfrak{p} \cong \mathbf{Z}/p\mathbf{Z}$ maps α to $(c \bmod p)$, and $\mathfrak{p}$ is generated, as an ideal, by p and $c - \alpha$. The map $\mathbf{Z}[\alpha] \to \mathbf{Z}[\alpha]/\mathfrak{p}$ can be used to test whether a given element of $\mathbf{Z}[\alpha]$ is contained in $\mathfrak{p}$: namely, one has $\sum_i s_i\alpha^i \in \mathfrak{p}$ if and only if $\sum_i s_i c^i \equiv 0 \bmod p$, with p, c as above.

Let a, b be coprime integers. Every prime ideal of $\mathbf{Z}[\alpha]$ that contains $a + b\alpha$ is a first degree prime ideal (see [26, Section 5, Lemma; 7, Corollary 5.5]), and as we just saw $a + b\alpha$ is contained in the prime ideal corresponding to $(p, c \bmod p)$ if and only if $a + bc \equiv 0 \bmod p$. This implies that the prime ideal factorization of $a + b\alpha$ corresponds to the prime factorization of its norm $a^d - t(-b)^d$, as follows. If $a^d - t(-b)^d$ contains the prime factor p exactly k times, with $k > 0$, then $a \equiv -bc \bmod p$ for a unique $c \bmod p$ for which $f(c) \equiv 0 \bmod p$, and the first degree prime ideal corresponding to $(p, c \bmod p)$ divides $a + b\alpha$ exactly to the kth power. So, one ideal of norm p accounts for the full exponent of p in $a^d - t(-b)^d$.

We denote by $\pi_\mathfrak{p}$ an element of $\mathbf{Z}[\alpha]$ that generates $\mathfrak{p}$. Such an element exists because $\mathbf{Z}[\alpha]$ is assumed to be a principal ideal domain; it is unique only up to multiplication by units. To pass from the prime ideal factorization of $a + b\alpha$ to its prime factorization it suffices to replace each prime ideal factor $\mathfrak{p}$ by $\pi_\mathfrak{p}$ and to multiply the result by a suitable unit.

2.7. *Step 1 of the number field sieve.* The discussion above leads to the following selection of the factor base. First select two smoothness bounds, B_1 and B_2. In

practice these bounds are best determined empirically. See Section 8 for some examples, and 6.2 and 6.3 for choices that are satisfactory from a theoretical point of view. We will use B_1 as smoothness bound for the integers $a+bm$, and B_2 as smoothness bound for the algebraic integers $a+b\alpha$. Now let $I = P \cup U \cup G$, where P is the set of all prime numbers $\leq B_1$, the set U is a set of generators for the group of units of $\mathbf{Z}[\alpha]$, and G consists of the elements $\pi_{\mathfrak{p}} \in \mathbf{Z}[\alpha]$, where $\mathfrak{p}$ ranges over the set of first degree prime ideals of $\mathbf{Z}[\alpha]$ of norm $\leq B_2$. The factor base is then formed by the elements $a_i = \varphi(i) \in \mathbf{Z}/n\mathbf{Z}$, for $i \in I$. We assume that $\gcd(a_i, n) = 1$ for $i \in I$; if that is not the case n can easily be factored, and the algorithm terminates.

To complete the description of Step 1 it remains to explain how to construct the sets U and G. This will be done in Section 3.

2.8. *Step 2 of the number field sieve.* We discuss how Step 2 is performed. First, select two additional bounds B_3 and B_4. These bounds are again best determined empirically. See 4.6 for various considerations concerning their choice, and Section 8 for examples. To find relations among the a_i, one searches for pairs of integers (a, b), with $b > 0$, satisfying the following conditions:

(i) $\gcd(a, b) = 1$;

(ii) $|a + bm|$ is B_1-smooth, except for at most one additional prime factor p_1, which should satisfy $B_1 < p_1 < B_3$;

(iii) $a+b\alpha$ is B_2-smooth, except for at most one additional prime ideal factor $\mathfrak{p}_2$, of which the norm p_2 should satisfy $B_2 < p_2 < B_4$.

We will assume that $a + bm > 0$; in the unlikely event that $a + bm < 0$, replace (a, b) by $(-a, -b)$.

The prime number p_1 in (ii) is called the *large prime*, and the additional prime ideal $\mathfrak{p}_2$ in (iii) the *large prime ideal*. Note that $\mathfrak{p}_2$ corresponds to the pair $(p_2, c \bmod p_2)$, where c is such that $a \equiv -bc \bmod p_2$; this enables us to distinguish between prime ideals of the same norm. If the large prime does not occur, then we write $p_1 = 1$. Likewise, if the large prime ideal does not occur in (iii), we write symbolically $\mathfrak{p}_2 = 1$ and $p_2 = 1$. Pairs (a, b) for which $p_1 = p_2 = 1$ will be called *full* relations, and the other pairs *partial* relations.

The search for pairs (a, b) satisfying (i), (ii), and (iii) above can be carried out by means of the sieving technique described in Section 4. We show how the pairs give rise to relations between the a_i. First, suppose that (a, b) is a full relation. By (ii), there is an identity of the form

$$a + bm = \prod_{p \in P} p^{e(p)},$$

with $e(p) \in \mathbf{Z}_{\geq 0}$. From (i), (iii), and 2.6 it follows that $a + b\alpha$ can be written as a product of elements $\pi_{\mathfrak{p}} \in G$ to certain powers, and a unit from $\mathbf{Z}[\alpha]$. One can determine the contribution from G by considering the factorization of $\mathbf{N}(a + b\alpha) = a^d - t(-b)^d$, as explained in 2.6. Since U generates the group of units of $\mathbf{Z}[\alpha]$, the unit contribution to the prime factorization of $a + b\alpha$ can be written

as a product of elements from U. In Section 5 it is explained how this is achieved. As a result, we get an identity of the form

$$a + b\alpha = \prod_{u \in U} u^{e(u)} \cdot \prod_{g \in G} g^{e(g)},$$

with $e(u) \in \mathbf{Z}$ and $e(g) \in \mathbf{Z}_{\geq 0}$. Because $a + bm$ and $a + b\alpha$ have the same image under φ, these two factorizations lead to the identity

$$\prod_{p \in P} \varphi(p)^{e(p)} = \prod_{u \in U} \varphi(u)^{e(u)} \cdot \prod_{g \in G} \varphi(g)^{e(g)} \tag{2.9}$$

in $\mathbf{Z}/n\mathbf{Z}$, from which one obtains a relation $v = (v_i)_{i \in I} \in \mathbf{Z}^I$ between the a_i by putting $v_i = e(i)$ for $i \in P$, and $v_i = -e(i)$ for $i \notin P$. Note that $a_i^{v_i}$ exists for $i \notin P$ because $\gcd(a_i, n) = 1$ for $i \in I$.

In this way each full relation (a, b) leads to a relation between the a_i as in (2.3).

2.10. *Making use of partial relations.* As we will see in Section 4, partial relations can be found at little extra cost during the search for full relations. Furthermore, they occur much more frequently than full relations, so that relatively many of them are found; so many, in fact, that there are quite a few with the same large prime or the same large prime ideal. If that occurs it may be possible to convert a collection of partial relations into a relation among the a_i as in (2.3), as follows.

A set C of partial relations is called a *cycle* if for each $(a, b) \in C$ there is a sign $s(a, b) \in \{-1, +1\}$ such that

$$\prod_{(a,b) \in C} (a + bm)^{s(a,b)} = \prod_{p \in P} p^{e(p)},$$

with $e(p) \in \mathbf{Z}$, and

$$\prod_{(a,b) \in C} (a + b\alpha)^{s(a,b)} = \prod_{u \in U} u^{e(u)} \cdot \prod_{g \in G} g^{e(g)},$$

with $e(u)$, $e(g) \in \mathbf{Z}$. Informally, this means that if a prime $p_1 > B_1$ occurs in the factorization of $a + bm$ for some pair $(a, b) \in C$, then p_1 also occurs in the factorization of $\bar{a} + \bar{b}m$ for some pair $(\bar{a}, \bar{b}) \in C$ with $s(\bar{a}, \bar{b}) = -s(a, b)$. Similarly, each occurrence of a large prime ideal in $a + b\alpha$ is canceled by the occurrence of the same large prime ideal in another pair with the opposite sign.

For each cycle one can compute the exponents $e(p)$ by adding or subtracting the exponents occurring in the factorizations of the $a + bm$ for the pairs (a, b) in the cycle, according to their signs. Similarly, one computes the $e(g)$ by adding or subtracting the exponents in the prime ideal factorizations of the $a + b\alpha$; these exponents are found as explained in 2.6. Once the $e(g)$ have been computed, the $e(u)$ are found with the method given in Section 5, cf. Remark 5.3.

Just as above, we now obtain the following relation between the a_i:

$$\prod_{p \in P} \varphi(p)^{e(p)} = \prod_{u \in U} \varphi(u)^{e(u)} \cdot \prod_{g \in G} \varphi(g)^{e(g)}.$$

This is the same as (2.9), except that now the integers $e(i)$ are allowed to be negative. Since $\gcd(a_i, n) = 1$ for $i \in I$, negative powers of the a_i are well defined.

In this way each cycle among partial relations leads to a relation between the a_i.

2.11. *Remark.* Partial relations for which $p_1 \neq 1$ but $p_2 = 1$ are referred to as *pf*'s (for 'partial-full'), because they would lead to an equation as in (2.9) with a partial factorization, i.e., a factor $\varphi(p_1)$, on the left hand side, and a full factorization on the right. Similarly, partial relations for which $p_1 = 1$ and $p_2 \neq 1$ are called *fp*'s, partial relations for which both $p_1 \neq 1$ and $p_2 \neq 1$ are called *pp*'s, and full relations are *ff*'s. We refer to 7.3 for a more general notion of partial relations.

The negative $s(a, b)$ in the cycles have the effect that the large primes and the large prime ideals in the resulting combinations are canceled. For cycles consisting of only two *pf*'s (with the same p_1) there is no need to make use of the signs $s(a, b)$, because a relation among the a_i of the form

$$x^2 \cdot \prod_{i \in I} a_i^{v_i} = 1$$

for some unit $x \in \mathbf{Z}/n\mathbf{Z}$ (of the form $\varphi(p_1)$), is just as useful as a relation like (2.3). However, doing the same for cycles involving *fp*'s or *pp*'s would introduce factors $\varphi(\pi_\mathfrak{p})$ for prime ideals $\mathfrak{p}$ of norm $> B_2$ into x, and would thus require finding generators of the large prime ideals involved in the cycles. This is avoided by means of the signs described above.

In 9.6 we shall encounter a variant of the number field sieve in which the use of generators is avoided, and in which the signs can be discarded. At the other extreme, 9.1 describes a variant in which *all* prime ideals are canceled, not just the large ones.

2.12. *Finding the cycles.* Write P_1 for the set of all large primes occurring in the partial relations, and P_2 for the set of all large prime ideals that occur. We view the set of partial relations as the set of edges of a graph with vertex set $\{1\} \cup P_1 \cup P_2$; namely, each partial relation with large prime p_1 and large prime ideal $\mathfrak{p}_2$ represents an edge between p_1 and $\mathfrak{p}_2$. The edges incident with 1 correspond to the *pf*'s and *fp*'s; except for these edges the graph is bipartite. Each cycle in the graph gives rise to a cycle among the partial relations. For cycles of even length one can assign the signs $+1$, -1 alternately to the partial relations corresponding to the edges. Cycles of odd length contain the vertex 1; again one can assign the signs alternately, but now starting with an edge incident with 1.

It is not necessary to find *all* cycles in the graph. For example, if the symmetric difference of two cycles C_1 and C_2 is a cycle C_3, then the relation between the

a_i obtained from C_3 is a linear combination of the relations obtained from C_1 and C_2. Therefore, if the cycles C_1, C_2 are already used, there is no point in using C_3 as well. In other words, it will suffice to find a maximal set of cycles that is "independent" in a suitable sense. In [28] it is explained how this can be done, and how a convenient representation for the graph can be constructed. Another way of dealing with the partial relations is to postpone their combination into cycles until Step 3, as discussed in 7.2.

2.13. *Free relations.* In addition to the relations that are based on full relations and cycles among partial relations, there are the *free* relations, which are much easier to come by. They are already valid in the ring $\mathbf{Z}[\alpha]$, before φ is applied. There is one such relation for each prime number $p \leq \min(B_1, B_2)$ for which the polynomial $f = X^d - t$ factors completely into linear factors modulo p. Namely, let p be such a prime, and write $X^d - t \equiv \prod_c (X-c)^{e_c} \bmod p$, where c ranges over a set of integers that are pairwise distinct modulo p, and where the multiplicities e_c are positive integers (they are equal to 1 if p does not divide dt). Each c is a zero of $f \pmod p$, and therefore gives rise to a first degree prime ideal $\mathfrak{p}$ of norm p, as explained in 2.6; for this $\mathfrak{p}$ we write $e(\mathfrak{p}) = e_c$. With this notation, the ideal generated by p is equal to the product of the ideals $\mathfrak{p}^{e(\mathfrak{p})}$, so p can be written as the product of the elements $\pi_{\mathfrak{p}}^{e(\mathfrak{p})}$ multiplied by a unit:

$$p = \prod_{u \in U} u^{e(u)} \cdot \prod_{\mathfrak{p},\, \mathfrak{N}\mathfrak{p}=p} \pi_{\mathfrak{p}}^{e(\mathfrak{p})} \qquad \text{with } e(u) \in \mathbf{Z}.$$

This gives the identity

$$\varphi(p) = \prod_{u \in U} \varphi(u)^{e(u)} \cdot \prod_{\mathfrak{p},\, \mathfrak{N}\mathfrak{p}=p} \varphi(\pi_{\mathfrak{p}})^{e(\mathfrak{p})},$$

which has the form (2.9). The integers $e(u) \in \mathbf{Z}$ can again be found by means of the method explained in Section 5. The density of the set of primes that split completely in this way is the inverse of the degree of the splitting field of f, which divides $d \cdot \phi(d)$ and is a multiple of $\operatorname{lcm}(d, \phi(d))$, with ϕ the Euler ϕ-function. For example, if $d = 5$ then one out of every twenty primes splits completely in this way; if $B_1 \approx B_2$ this means that one may expect approximately one fortieth of all relations to come for free.

This completes the description of Step 2, and thereby the description of the number field sieve.

3. Finding generators

In this section we discuss the computation of the sets U and G introduced in 2.7.

An element of $\mathbf{Z}[\alpha]$ is a unit if and only if it has norm ± 1. The structure of the group of units can be described as follows. Let the polynomial f that was selected in 2.5 have r_1 real roots and $2r_2$ non-real complex roots, so that $r_2 = (d - r_1)/2$. Since f is of the form $X^d - t$, we have $r_1 = 1$ if d is odd; and if d is even, then $r_1 = 0$ if $t < 0$ and $r_1 = 2$ if $t > 0$. Let $l = r_1 + r_2 - 1$. With

this notation, the group of units of $\mathbf{Z}[\alpha]$ is generated by a suitable root of unity u_0 and l multiplicatively independent units $u_1, u_2, \ldots, u_l$ of infinite order; we may take $u_0 = -1$ if $r_1 > 0$. We shall let U consist of such elements $u_0, \ldots, u_l$.

Before we compute G we make a list of all first degree prime ideals of norm $\leq B_2$. As we saw in 2.6, this amounts to making a list of all pairs $(p, c \bmod p)$, where p is a prime number $\leq B_2$, and $c \in \mathbf{Z}$ satisfies $f(c) \equiv 0 \bmod p$. To find these pairs efficiently, one can use a probabilistic root finder for polynomials over finite fields, cf. [21, Section 4.6.2]; the number of pairs thus found, i. e., $\#G$, can be expected to be close to $\pi(B_2)$, the number of primes up to B_2. An element of $\mathbf{Z}[\alpha]$ generates $\mathfrak{p}$ if and only if it belongs to $\mathfrak{p}$ and has norm $\pm p$; in other words, the conditions to be met by $\pi_\mathfrak{p} = \sum_{i=0}^{d-1} s_i \alpha^i$ are that $\sum_{i=0}^{d-1} s_i c^i \equiv 0 \bmod p$ and $\mathbf{N}(\pi_\mathfrak{p}) = \pm p$. To determine G it suffices to find one such element for each pair $(p, c \bmod p)$.

In practice the search for elements of U and G is best carried out simultaneously. This can be done as follows. Fix a multiplier bound M and a search bound C, depending on K and B_2. We refer to 3.6 for a discussion of feasible choices of M and C. For the moment, one may think of M as a fairly small integer—in all cases that we did M could be taken less than 10—and of C as roughly proportional to $B_2^{2/d}$. The actual asymptotics are a little different, though.

For all first degree prime ideals $\mathfrak{p}$ for which we want to find a generator, put $m(\mathfrak{p})$ equal to $M + 1$. This number $m(\mathfrak{p})$ keeps track of the status of $\mathfrak{p}$ during the search process: if $m(\mathfrak{p}) > M$ no generator has been found yet, otherwise an element $\bar{\pi}_\mathfrak{p}$ of $\mathfrak{p}$ has been found with $\mathbf{N}(\bar{\pi}_\mathfrak{p}) = \pm m(\mathfrak{p})p$, where $p = \mathfrak{N}\mathfrak{p}$; then the ideal generated by $\bar{\pi}_\mathfrak{p}$ is $\mathfrak{p}$ times an ideal of norm $m(\mathfrak{p})$.

3.1. *Search algorithm.* For all $\gamma = \sum_{i=0}^{d-1} s_i \alpha^i \in \mathbf{Z}[\alpha]$ for which $\sum_{i=0}^{d-1} s_i^2 |\alpha|^{2i} \leq C$, compute the norm $\mathbf{N}(\gamma)$, cf. Remark 3.3; here $|\alpha|$ denotes the real number $|t|^{1/d}$. If $\mathbf{N}(\gamma)$ is of the form kp for some prime p from the list of pairs $(p, c \bmod p)$ and some non-zero integer k with $|k| \leq M$, do the following. Identify the first degree prime ideal $\mathfrak{p}$ that corresponds to this p and γ, in other words, the pair $(p, c \bmod p)$ for which $\sum_{i=0}^{d-1} s_i c^i \equiv 0 \bmod p$, and update the data concerning $\mathfrak{p}$: if $m(\mathfrak{p}) > |k|$ then replace $m(\mathfrak{p})$ by $|k|$ and put $\bar{\pi}_\mathfrak{p}$ equal to γ.

After all these γ have been processed, the $m(\mathfrak{p})$ are all $\leq M$ if the multiplier bound M and the search bound C have been chosen properly. For the $\mathfrak{p}$ with $m(\mathfrak{p}) = 1$, put $\pi_\mathfrak{p}$ equal to $\bar{\pi}_\mathfrak{p}$. For the other $\mathfrak{p}$, compute $\pi_\mathfrak{p}$ by dividing $\bar{\pi}_\mathfrak{p}$ by a generator of the appropriate ideal of norm $m(\mathfrak{p})$. This requires the computation of generators of the ideals of norm at most M, as well as the inverses of these generators. There are only a few such ideals, and generators for them are often easy to find; in general, one may hope to encounter them during the search just described. If this doesn't work, one may have to appeal to one of the methods indicated in 3.8 below.

During the same search one keeps track of the units that are encountered. These are not only the elements γ with $\mathbf{N}(\gamma) = \pm 1$ that are found, but also quotients of two elements that have the same norm (up to sign) and that generate the same ideal; in the latter case a division is needed. Multiplicative dependencies

can be cast out with the help of the function ν from Section 5. The set U of units that we are left with will often be the set of l multiplicatively independent elements that we are looking for. If later in the algorithm it is discovered that the resulting set U does not generate the group of units of $\mathbf{Z}[\alpha]$, then this discovery leads to a new unit, which can be used to alter U; see Remark 5.4.

3.2. *Remark.* If $r_1 > 0$ it is useful to require that all elements of G and U except $u_0 = -1$ are positive under some particular embedding of $\mathbf{Q}(\alpha)$ into $\mathbf{R}$. For this purpose one fixes one particular real embedding, and one replaces x by $-x$ for each $x \in G \cup U$, $x \neq -1$ that is negative under this embedding.

3.3. *Remark.* For $\gamma = \sum_{i=0}^{d-1} s_i \alpha^i$ the norm $\mathbf{N}(\gamma)$ is a homogeneous dth degree polynomial in the s_i with coefficients that are integers depending on the polynomial f. In our implementation of Algorithm 3.1 the norm-polynomial was 'hard-wired', i.e., each new f required changes in the program and thus recompilation. Furthermore, the search was organized in such a way that the norm of each γ was obtained from the norm of the previous γ by just a few arithmetic operations. This greatly enhanced the speed of our searching program.

In the rest of this section we discuss a few technical difficulties related to the search for U and G. Some of these difficulties were actually encountered during the factorizations reported in Section 8. Some others we did not encounter, but we can vividly imagine that this will happen to others who try the algorithm. Finally, there are difficulties of a primarily theoretical nature, which come up when one attempts to analyse the run time of the algorithm. We resolve these difficulties by using the tools that have been developed in algorithmic algebraic number theory. For general background, see [43; 8; 30; 33].

3.4. *Lack of unique factorization.* In the description of the algorithm we made the assumption that $\mathbf{Z}[\alpha]$ is a unique factorization domain. This is a strong assumption, which indeed fails to hold in three of the examples given in Section 8. The assumption implies that $\mathbf{Z}[\alpha]$ is equal to the ring of integers of K, which in turn implies that $e \equiv 0$ or $-1 \bmod d$. We now discuss how to proceed if $\mathbf{Z}[\alpha]$ is not assumed to have unique factorization. We note that in general it is not easy to check whether $\mathbf{Z}[\alpha]$ has unique factorization, but if desired this can be done along the way.

3.5. *The ring of integers.* One starts by replacing $\mathbf{Z}[\alpha]$ by the ring A of algebraic integers in K. Methods for determining A can be found in the references just given; see also [5]. The discriminant of f, which equals $\pm d^d t^{d-1}$, can for bounded r and $|s|$ easily be factored into primes, and once this prime factorization is available the determination of A proceeds in time $(d+\log|t|)^{O(1)}$. A few examples of rings of integers A are given in 2.5 and in Section 8. If $\mathbf{Z}[\alpha] \neq A$ then $\mathbf{Z}[\alpha]$ is not a unique factorization domain.

We shall denote the absolute value of the discriminant of A by Δ. This number is given by $\Delta = d^d|t|^{d-1}/[A : \mathbf{Z}[\alpha]]^2$, and it is usually determined simultaneously with A. One can show that Δ divides $d^d(r|s|)^{d-1}$ and that it is at least $d^d/(1+\log d)^{cd}$ for some absolute positive constant c.

The replacement of $\mathbf{Z}[\alpha]$ by A necessitates a few modifications to the algorithm. The first is that the ring homomorphism $\varphi\colon \mathbf{Z}[\alpha] \to \mathbf{Z}/n\mathbf{Z}$ needs to be extended to A. This can be done if the natural condition $\gcd(drs, n) = 1$ is satisfied. Namely, any element $\gamma \in A$ can be written as $\gamma = \beta/m$, where $m \in \mathbf{Z}$ is built up from prime numbers dividing drs; then $\varphi(m)$ has an inverse in $\mathbf{Z}/n\mathbf{Z}$, and we can extend φ to a ring homomorphism $A \to \mathbf{Z}/n\mathbf{Z}$ by putting $\varphi(\gamma) = \varphi(\beta)\varphi(m)^{-1}$.

Secondly, it is, for the ring A, not necessarily true that each prime ideal $\mathfrak{p}$ dividing an expression $a + b\alpha$, with $a, b \in \mathbf{Z}$ coprime, is a first degree prime ideal in the sense that $\#A/\mathfrak{p}$ is a prime number. In addition to the first degree prime ideals, one may also encounter prime ideals $\mathfrak{p}$ that contain a prime number p dividing the index $[A : \mathbf{Z}[\alpha]]$ of additive groups and that intersect $\mathbf{Z}[\alpha]$ in a first degree prime ideal of $\mathbf{Z}[\alpha]$; such p divide drs. In the rest of this section we call these prime ideals *exceptional*. In order to compute the prime ideal factorizations of the expressions $a + b\alpha$ one needs to construct the exceptional prime ideals $\mathfrak{p}$ as well as the corresponding valuations. The existence of an efficient algorithm for doing this follows from [30, Theorem 4.9, and the discussion following its proof]; often there are faster *ad hoc* ways to proceed than the one indicated in [30].

3.6. *Searching for prime elements.* Denote by ω_d the volume of the unit ball in $\mathbf{R}^d$. We have $\omega_d = \pi^{d/2}/\Gamma(1 + \frac{d}{2})$, where $\Gamma(1 + \frac{d}{2})$ can be calculated from $\Gamma(1 + z) = z\Gamma(z)$, $\Gamma(1) = 1$, and $\Gamma(\frac{1}{2}) = \sqrt{\pi}$. Next put $v_d = (4/d)^{d/2}/\omega_d$, which is $((2 + o(1))/(\pi e))^{d/2}$ for $d \to \infty$, and

$$C = (v_d \cdot \sqrt{\Delta} \cdot B_2)^{2/d}, \qquad M = [v_d \cdot \sqrt{\Delta}].$$

With this notation, one searches among all non-zero elements $\gamma \in A$ for which $\frac{1}{d}\sum_\sigma |\sigma\gamma|^2 \leq C$, with σ ranging over the embeddings $K \to \mathbf{C}$; if we write $\gamma = \sum_{i=0}^{d-1} q_i \alpha^i$ with $q_i \in \mathbf{Q}$, then $\frac{1}{d}\sum_\sigma |\sigma\gamma|^2 = \sum_{i=0}^{d-1} q_i^2 |t|^{2i/d}$. One of the purposes of the search is to find, for each first degree or exceptional prime ideal $\mathfrak{p}$ with $M < \mathfrak{N}\mathfrak{p} \leq B_2$, a non-zero element $\pi_\mathfrak{p} \in \mathfrak{p}$ with $|\mathbf{N}(\pi_\mathfrak{p})| \leq M\mathfrak{N}\mathfrak{p}$. One can check whether a given element γ can play the role of $\pi_\mathfrak{p}$ for some $\mathfrak{p}$ if one knows the norm of γ, as in 3.1.

It is a consequence of the Minkowski lattice point theorem that at the end of the search for each $\mathfrak{p}$ an element $\pi_\mathfrak{p}$ has been found. The ideal generated by $\pi_\mathfrak{p}$ is then not necessarily equal to $\mathfrak{p}$, but it is equal to $\mathfrak{p}$ multiplied by an ideal of norm at most M. From $\mathfrak{N}\mathfrak{p} > M$ one sees that $\mathfrak{p}$ occurs exactly once in $\pi_\mathfrak{p}$.

The theory of sphere packings implies that the choice of v_d above is not the best one, and in practice one would do wise to experiment with smaller values for v_d. For the purposes of a complexity analysis the value given above is good enough, since any feasible choice for v_d is outweighed by $\sqrt{\Delta}$.

3.7. *Factoring elements.* We shall write G' for the set of $\pi_\mathfrak{p}$'s found in 3.6, with $\mathfrak{p}$ ranging over the set of first degree or exceptional prime ideals for which $M < \mathfrak{N}\mathfrak{p} \leq B_2$. In addition, we write H for the multiplicative group of non-zero elements $\gamma \in K$ for which the fractional ideal $A\gamma$ is built up from the prime

ideals of norm $\leq M$. All units of A clearly belong to H. The set G' and the group H will play the role that the set G and the group of units of $\mathbf{Z}[\alpha]$ played in Section 2. For example, the algorithm of Section 2 requires that we write certain expressions of the form $a + b\alpha$ as a product of powers of elements of G and a unit, and similarly for certain alternating products of such expressions. In the modified algorithm, we write expressions of the same type as a product of elements of G' and an element of H. To determine which power of $\pi_{\mathfrak{p}} \in G'$ occurs in a given expression one proceeds exactly as in Section 2, using 2.6, except if $\mathfrak{p}$ is exceptional; in the latter case one needs to apply the valuations mentioned in 3.5.

3.8. *Generators for H.* The next step is to find a multiplicative representation for the elements of H, by means of a set U' that plays the role of U. One can attempt to find such a set U' by means of the method indicated in 3.1. Namely, during the search in 3.6 one also keeps track of elements that are entirely built up from prime ideals of norm at most M. If one is lucky, one obtains in this way not only a set U of generating units, but also generators for each of the prime ideals of norm at most M, either directly or by combining a few elements that are found. The set U' then consists of U together with the generators of those prime ideals, and as in 3.1 one can modify the elements $\pi_{\mathfrak{p}} \in G'$ found in 3.6 so as to obtain true generators for the larger prime ideals $\mathfrak{p}$. Altogether this situation is very similar to what we had earlier, the main difference being that $\mathbf{Z}[\alpha]$ is replaced by A. In particular, A is a unique factorization domain in this case.

In the examples that we tried the above is, essentially, what happened. In general one cannot expect to be so fortunate. In the first place, the ring A need not be a unique factorization domain, in which case there do not exist generators for all primes of norm at most M. A more serious difficulty is caused by the possibility that A does not have a set of "small" generating units. If this occurs, then not only the units, but also generators of some of the prime ideals may be hard to find. It seems likely that this difficulty may actually be encountered in practice. Since we have no experience with it, we do not know which of our ideas for dealing with it is to be recommended for practical use. We shall just make a few remarks of a theoretical nature, which indicate that there do exist satisfactory ways to solve the problem. In this discussion we make no assumptions on unique factorization in A or about the units of A.

One possibility is to use an algorithm of Buchmann, see [4; 30, Theorem 5; 6, Section 6]. If properly modified and interpreted, this algorithm yields a set of independent generators U' for the group H, and these generators are such that there is a fast algorithm that given $\gamma \in H$ finds the unique expression of γ as a product of powers of elements of U'. This means that U' can in a satisfactory way play the role of U.

A second possibility is to use Theorem 6.2 of [30] in order to find a set of generators for H. In order to convert this set of generators into an independent set U' for which a fast algorithm as just indicated exists, one can apply linear algebra over $\mathbf{Z}$ (see [20]) as well as basis reduction techniques (cf. [18]).

A third possibility is not to bother about finding a generating set for H at all, but waiting for elements of H to produce themselves in the course of the algorithm. For example, every B_2-smooth expression $a + b\alpha$ that is found gives, upon division by an appropriate product of elements of G', an element of H, and likewise for the alternating products of expressions $a + b\alpha$ that arise from the partial relations. The collection of all elements of H that produce themselves in this way generates a subgroup H' of H. It is not guaranteed that H' equals H, but since we never encounter elements of H outside of H' this is of no concern to us. We do need to convert the given set of generators for H' into a set U' of independent generators for H'. This can be done as indicated above. See also Remark 5.4.

Once a suitable set U' has been found, it is necessary to find, for each $u \in U'$, an element of $\mathbf{Z}/n\mathbf{Z}$ that can meaningfully be called $\varphi(u)$. If U' is contained in A this presents no problem, since φ is defined on A. If U' is not contained in A we can proceed as we did with A itself; this can be done if n is free of prime factors $\leq M$, a condition that can easily be checked.

It is to be remarked that the third possibility for finding U' mentioned above can in principle be extended to G': do not search for elements $\pi_{\mathfrak{p}}$, but wait for them (or for elements that are just as good) to produce themselves in the course of the algorithm. This approach might be feasible for number fields for which Δ is much larger, and it might therefore be useful if one wants to apply the number field sieve to arbitrary positive integers n. The resulting algorithm is, in a somewhat different formulation, discussed in 9.1. As we shall see in 9.6, there is also a variant of the number field sieve that dispenses with G' and U' altogether.

In several of the manipulations with elements of H and G' that we just sketched it happens that the elements u that one is interested in arise as the product of powers of certain other elements and their inverses. In this case it may be laborious to calculate the explicit expression $u = \sum_{i=0}^{d-1} q_i \alpha^i$ of u in terms of the powers of α. It is good to keep in mind that this calculation can usually be avoided. This is because the information on u that the algorithm really needs—such as the vector $\nu(u)$ defined in Section 5, the argument of u under one particular embedding $K \to \mathbf{C}$, and the value of $\varphi(u) \in \mathbf{Z}/n\mathbf{Z}$—can all be derived from the given product representation for u.

4. Sieving

In this section we describe how the search for full and partial relations can be carried out. In the notation of Section 2, these relations correspond to pairs of coprime integers (a, b) such that $a + bm$ is B_1-smooth, except for at most one prime factor $< B_3$, and such that $a + b\alpha$ is B_2-smooth, except for at most one prime ideal of norm $< B_4$.

From a theoretical point of view one can solve the problem of finding these pairs (a, b) by applying the elliptic curve smoothness test [25, Section 4.3] to each individual pair, because smoothness of the algebraic integer $a + b\alpha$ is equivalent to smoothness of the integer $a^d - t(-b)^d$, cf. 2.6. In this section we present a more

practical way to find the pairs (a, b). For the purposes of the run time analysis in Section 6 the two methods are equivalent.

We describe a method to find pairs (a, b) that satisfy the conditions (i), (ii), and (iii) of 2.8 for some fixed positive value of b, and for a ranging over an interval $[a_{\min}, a_{\max})$. This method is applied to all b in $[1, b_{\max}]$ that are to be processed. There is no particular order in which this has to be done. We shall see, however, that one can gain some efficiency by processing the b's in order. This is to be kept in mind if the search for relations is carried out in parallel on many independent processors: it is better to assign a range of consecutive b-values to each processor than some arbitrary set of b-values. An entirely different way to organize the sieving step is described in [2].

The values of $a_{\min}$ and $a_{\max}$ are best determined empirically; see Sections 6 and 8 for theoretical and practical choices. It is not necessary to make a choice for $b_{\max}$, since one can simply continue until the number of full relations plus the number of independent cycles among the partial relations is larger than $\#I$, the cardinality of the factor base. With growing b, however, the probability that both $a + bm$ and $a + b\alpha$ are smooth gets smaller, and quite noticeably so. This means that if B_1 and B_2 have been chosen too low, then one might *never* find sufficiently many relations. See Section 8 for examples of $b_{\max}$ and Section 6 for a theoretical estimate.

4.1. *Two sieves.* Fix some positive value for b. Testing the numbers $a + bm$ with $a \in [a_{\min}, a_{\max})$ for B_1-smoothness can be done by means of a sieve over a, because p divides $a + bm$ for all a that are $-bm \bmod p$. After sieving with all $p \leq B_1$, one can identify the pairs (a, b) that have a reasonable chance to satisfy 2.8(ii), and one has to inspect the corresponding $a+b\alpha$'s for B_2-smoothness, cf. 2.8(iii). If there are only a few candidates, one can do this using trial division of the norms $N(a + b\alpha)$ (after checking that 2.8(i) holds), cf. 2.6. In practice it will be much faster to apply a second sieve, again over the entire interval of a-values, because the number of candidates, for a proper choice of the factor base, will be considerable. A sieve can be applied because the first degree prime ideal corresponding to a pair $(p, c \bmod p)$, as in 2.6, occurs in $a + b\alpha$ for all a that are $-bc \bmod p$.

Only pairs (a, b) for which both $a + bm$ (after the first sieve) and $a + b\alpha$ (after the second sieve) are likely to satisfy the smoothness conditions of 2.8(ii) and 2.8(iii), are subjected to further gcd and trial division tests to see if the pair indeed gives rise to a full or a partial relation. The rest of this section is devoted to a more detailed description of a possible implementation of the sieving step.

4.2. *The rational sieve.* We describe how the $a + bm$, for some fixed b, can be sieved for B_1-smoothness. For all $a \in [a_{\min}, a_{\max})$ initialize the sieve locations s_a as zero. Next, for all primes $p \leq B_1$, replace s_a by $s_a + \log p$ for all $a \in [a_{\min}, a_{\max})$ that are $-bm \bmod p$.

If, after all primes $p \leq B_1$ have been processed, a sieve location s_a is close to $\log |a + bm|$, then it is quite likely that $a + bm$ is B_1-smooth. If $s_a \geq \log(a + bm) - \log B_3$, then $a + bm$ is probably B_1-smooth except for a factor that is at most B_3; if $B_3 < B_1^2$, then this factor will be prime, if we assume that $a + bm$

is square-free. The event that $s_a \geq \log(a + bm) - \log B_3$ is called a *report* (but see 4.3).

4.3. *Efficiency considerations.* Because we do not sieve with prime powers, not all smooth $a + bm$ are caught in the sieve: numbers $a + bm$ that are smooth and not square-free may be overlooked. This is only a first step in speeding up the sieving without affecting its yield by too much. In practice quite a few more smooth $a + bm$ will be missed, because 4.2 is only an idealized version of what actually happens. The s_a, for instance, are usually represented by 8-bit (1-byte) integers. Consequently, $\log p$ is rounded to the nearest integer, and the base of the logarithm is chosen large enough so that overflow is avoided when 8-bit (or 7-bit, see below) integers are added. Furthermore, one often does not sieve with the small primes below a certain small prime bound, or one replaces them by a small power. This means that one should use $\log(a + bm) - \log B_3 - B_5$ instead of $\log(a + bm) - \log B_3$ while checking for reports, for some B_5 that depends on the small prime bound. The small prime bound and the corresponding B_5 are best determined empirically.

It is often a good idea to allow negative s_a, i. e., 7-bit integers plus one bit for the sign. This makes it possible to initialize the s_a as $-\log(a+bm)+\log B_3+B_5$ so that the report-check can be replaced by a non-negativity check, which is often faster. In many architectures four consecutive 8-bit s_a's can be checked simultaneously for non-negativity by means of one 32-bit 'and'-operation with the proper mask. Since a will be small compared to bm, all s_a can be initialized to the same rounded value $-\log(bm)+\log B_3+B_5$. This often allows a simultaneous initialization of several consecutive s_a's. All these changes are intended to decrease the cost of the sieving step, while some of them have a negative effect on the performance. Care should be taken that the cost/performance ratio does *not* increase.

The computation of $-bm \bmod p$ requires a (multi-precision) division by p, unless $-(b-1)m \bmod p$ is known, in which case a few additions and comparisons suffice. This makes consecutive processing of the b's slightly faster than processing them in random order. A similar remark applies to the second, algebraic sieve.

4.4. *The algebraic sieve and trial division.* We describe how to process the reports from 4.2 in order to locate the pairs (a, b) that satisfy conditions 2.8(i), (ii), and (iii), for the same fixed b as in 4.2. The improvements of 4.3 are taken into account.

First, check for reports: let $A = \{a : s_a \geq 0\}$ be the set of a's at which reports occur, replace s_a for $a \in A$ by some moderately small negative number B_6, and leave the other s_a unchanged. Next use a sieve to replace, for all first degree prime ideals $\mathfrak{p}$ with $\mathfrak{N}\mathfrak{p} \leq B_2$, the number s_a by $s_a + \log p$ for all $a \in [a_{\min}, a_{\max})$ with $a \equiv -bc \bmod p$, where $(p, c \bmod p)$ corresponds to $\mathfrak{p}$.

Finally, do the following for all $a \in A$ for which $s_a \geq 0$. If $\gcd(a, b) = 1$, then compute $\log|a^d - t(-b)^d|$—for which a crude floating point approximation to the integer $a^d - t(-b)^d$ suffices—and check if $s_a \geq \log|a^d - t(-b)^d| + B_6 - \log B_4$, cf. 2.6. If that turns out to be the case, attempt to factor $|a + bm|$

using trial division by the primes $\leq B_1$. If $|a + bm|$ is B_1-smooth, except for at most one prime factor $< B_3$, compute $|a^d - t(-b)^d|$, and attempt to factor this number using trial division by the primes $\leq B_2$. If $|a^d - t(-b)^d|$ turns out to be B_2-smooth, except for at most one prime factor $< B_4$, then a pair (a, b) satisfying 2.8(i), (ii), and (iii) has been found.

We introduced B_6 because $|a^d - t(-b)^d|$ can vary considerably over the a-interval. This implies that no uniform negative initialization of the s_a as in 4.3 can be used, which would allow an easy non-negativity check after the sieving. To avoid the computation of $\gcd(a, b)$ (and of $\log|a^d - t(-b)^d|$) for all $a \in A$ (with $\gcd(a, b) = 1$), we initialize the s_a for $a \in A$ as B_6, and we only compute the corresponding gcd (and possibly the logarithm) if s_a has at least made it to a non-negative number after the sieving. This saves some time, but it also introduces an extra inaccuracy, because values of a close to a zero of $a^d - t(-b)^d$ can be overlooked if B_6 is chosen too small. The value for B_6 is best determined empirically. Notice that overflow may occur in s_a for $a \notin A$, which one can avoid by changing these s_a to the smallest negative value that they can assume instead of leaving them unchanged during the report-check.

One can also compute $\gcd(a, b)$ before putting a in A. In that way fewer locations have to be checked after the second sieve. On the other hand, many more gcd's would have to be computed than in the version given above, because in that version the $a \in A$ for which $s_a < 0$ after the second sieve are cast out. It depends on the relative speed of the various operations which version is preferable.

4.5. *Remark.* If $a_{\max} - a_{\min}$, the number of memory locations needed for the sieve, is more than can be allocated, then the interval $[a_{\min}, a_{\max})$ should be partitioned into subintervals to which 4.2 and 4.4 can be applied. If the subintervals are processed in order (of the a's), then one can easily arrange an efficient transition from one subinterval to the next, by remembering the last visited a-value for each p and $\mathfrak{p}$.

4.6. *Choice of B_3 and B_4.* We conclude this section with a few remarks concerning the choice of the large prime bounds B_3 and B_4. In the theoretical analysis in Section 6 the partial relations play no role, cf. 6.2 and 6.5. This implies that the choices $B_3 = B_1$ and $B_4 = B_2$, for which all relations are full relations, are good enough from a theoretical point of view. In practice, however, partial relations make the algorithm run substantially faster, as can be seen in Section 8. The choice of B_3 and B_4 depends on various considerations. In the first place, one has to choose them such that $B_3 < B_1^2$ and $B_4 < B_2^2$ to avoid the problem of having to factor the remaining factor of $|a + bm|$ or $|a^d - t(-b)^d|$ after trial division by the primes $\leq B_1$ or $\leq B_2$, respectively. Large choices within the respective ranges result in many reports, a slow performance of the sieving step, but a very high yield. This may sound good, but large primes near the lower end of the range have a higher probability to be matched in a cycle, whereas the majority of the other partial relations will turn out to be useless, after having slowed down the sieving step and clogged up the disks. A reasonable choice for B_3 seems to be somewhere between $B_1^{1.2}$ and $B_1^{1.4}$, and similarly for B_4.

5. Finding the unit contribution

Let the notation be as in Sections 2 and 3, and let a, b be coprime integers for which $a + b\alpha$ is B_2-smooth. Then the ideal generated by $a + b\alpha$ can be written as the product of first degree prime ideals $\mathfrak{p}$ of norm $\leq B_2$:

$$(a + b\alpha) = \prod \mathfrak{p}^{e(\mathfrak{p})}.$$

In 2.6 we saw how the $e(\mathfrak{p}) \in \mathbf{Z}_{\geq 0}$ can be determined. The element $a + b\alpha$ itself can now be written as

$$a + b\alpha = \prod_{u \in U} u^{e(u)} \cdot \prod_{g \in G} g^{e(g)}, \tag{5.1}$$

where $e(g) = e(\mathfrak{p})$ if $g = \pi_{\mathfrak{p}}$, and with integers $e(u)$ that remain to be found.

Of course, one can find the unit $\prod_{u \in U} u^{e(u)}$ by computing $(a+b\alpha)\cdot\prod_{g \in G} g^{-e(g)}$ in the number field. Given a sufficiently large table of products of elements of U and their inverses, the $e(u)$ can then be found by table look-up. For very "small" fields this will probably work quite satisfactorily. However, when the field is a bit "larger", it will be quite slow, due to the arithmetic in the number field, which consists of fairly expensive polynomial multiplications and divisions modulo f. In this section we describe a faster method for determining the $e(u)$. The method keeps track of some extra information per generator $g \in G$, and uses vector additions instead of arithmetic in the number field.

Let $U = \{u_0, u_1, \ldots, u_l\}$ be as in Section 3. Choose l embeddings φ_1, φ_2, $\ldots$, φ_l of K into $\mathbf{C}$ such that no two of the φ_i are complex conjugates. This can be done as follows. Let f have r_1 real roots α_1, α_2, $\ldots$, α_{r_1}, and $2r_2 = d - r_1$ non-real complex roots α_{r_1+1}, α_{r_1+2}, $\ldots$, α_d, with $\alpha_{r_1+r_2+j}$ the complex conjugate of α_{r_1+j}, for $1 \leq j \leq r_2$. In Section 3 we saw that $l = r_1 + r_2 - 1$. For $1 \leq i \leq l$ let φ_i be the embedding $K \longrightarrow \mathbf{C}$ that maps $\sum_{j=0}^{d-1} q_j \alpha^j$ to $\sum_{j=0}^{d-1} q_j \alpha_i^j$.

For $x \in K$, $x \neq 0$, let $\nu(x)$ be the l-dimensional real vector with ith coordinate equal to $\log|\varphi_i(x)| - (\log|\mathbf{N}(x)|)/d$, for $1 \leq i \leq l$; if x is taken to be a unit of $\mathbf{Z}[\alpha]$ then $|\mathbf{N}(x)| = 1$ so that the term $-(\log|\mathbf{N}(x)|)/d$ can be omitted. Note that $\nu(u_0) = 0$. Let W be the $l \times l$ matrix having $\nu(u_i)$ as its ith column, for $1 \leq i \leq l$. The image of the group of units of $\mathbf{Z}[\alpha]$ under ν is a lattice in $\mathbf{R}^l$, a basis for this lattice being given by the columns of W.

To find the $e(u) \in \mathbf{Z}$ that satisfy (5.1) we notice in the first place that $v = (a + b\alpha) \cdot \prod_{g \in G} g^{-e(g)}$ is a unit, and that

$$\nu(v) = \nu(a + b\alpha) - \sum_{g \in G} e(g) \cdot \nu(g). \tag{5.2}$$

Since v is a unit, $\nu(v)$ is in the lattice spanned by the columns of W, and more in particular $\nu(v) = W \cdot (e(u_1), e(u_2), \ldots, e(u_l))^T$, so that the $e(u_i)$ for $1 \leq i \leq l$ are the entries of the vector $W^{-1} \cdot \nu(v)$ and can be computed as such.

It remains to determine $e(u_0)$. If f has at least one real root, then we took $u_0 = -1$ in Section 3, and furthermore we specified some particular real embedding

such that the other u_i and all $g \in G$ are positive under this embedding, see Remark 3.2. So, we put $e(u_0) = 0$ if $a + b\alpha$ is positive under this embedding, and $e(u_0) = 1$ otherwise. If f has no real roots, then u_0 is some root of unity. In this case, choose some particular complex embedding, and select $e(u_0)$ in such a way that the arguments (angles) of this complex embedding of the left and right hand sides of (5.1) match.

In practice the mapping ν and the entries of the matrix W^{-1} are only computed in limited precision, and the entries of the vector $W^{-1} \cdot \nu(v)$ are rounded to integers. To avoid problems with limited precision computations, it helps to select (or to change) U such that the columns of W form a reduced basis for the lattice that they span. It also helps to select (or to change) the elements $g \in G$ such that the coordinates of $W^{-1} \cdot \nu(g)$ lie between $-\frac{1}{2}$ and $\frac{1}{2}$; one can achieve this by multiplying g by an appropriate product of units (to be determined with the help of ν). In our implementation, the vectors $W^{-1} \cdot \nu(g)$, for $g \in G$, were computed once and for all and kept in a file.

5.3. *Remark.* The same method can be applied to determine the unit contribution in a cycle C, where we are dealing with $\prod_{(a,b)\in C}(a + b\alpha)^{s(a,b)}$ instead of $a + b\alpha$ in (5.1): simply replace $\nu(a + b\alpha)$ in (5.2) by $\sum_{(a,b)\in C} s(a,b) \cdot \nu(a + b\alpha)$. The case of the free relations (see 2.13) is even easier, since $\nu(p)$ is the zero vector.

5.4. *Remark.* If the set U as determined in Section 3 fails to generate the unit group, or if one decides not to bother determining U at all (cf. 3.1 and 3.8), then the algorithm described in this section needs to be modified in the sense that the units must be processed as they come along. At each stage, one needs to keep track of a set of independent generators for the group generated by the units that have appeared so far. If a unit u is encountered for which $\nu(u)$ does not belong to the lattice spanned by the images, under ν, of the current set of generators, then this set of generators needs to be updated. This can be done by means of lattice basis reduction techniques. It is likely that only a few of such updates will be necessary. In the cases that we tried difficulties of this nature did not arise. If they do arise, then the remark made at the end of Section 3 may help to minimize the arithmetic that needs to be done with the units.

6. Run time analysis

In this section we present a heuristic estimate for the run time of the number field sieve. Currently there are several factoring algorithms that have a subexponential expected run time, such as the continued fraction algorithm, the quadratic sieve algorithm and its variants, the elliptic curve method, the number field sieve, Dixon's random squares algorithm, Vallée's two-thirds algorithm, and the class group relations method. Only for the last three algorithms has a rigorous analysis of the expected run time been given [31; 35; 41]. For the other algorithms the only available run time analysis is based on heuristic estimates, but in practice they perform better than the rigorously analyzed ones.

Each of the algorithms mentioned generates, implicitly or explicitly, a sequence of integers of which only the smooth ones are useful. Depending on the

algorithm and on its implementation, these integers are constructed deterministically or drawn from a certain distribution. In all cases, the expected number of smooth integers in the sequence plays an important role in the run time analysis. A satisfactory estimate for this expected number can be given if the integers are independently drawn from the uniform distribution on the interval $[1, B]$, for some upper bound B. However, none of the algorithms that we mentioned satisfies this condition. To obtain a heuristic analysis, one simply *assumes* that the smoothness probabilities are the same as in the independent, uniform case. Only for the random squares algorithm, the two-thirds algorithm, and the class group relations method has this actually been *proved*, and this leads to a rigorous analysis of their expected run times.

For the other algorithms, including the algorithm described in this paper, nothing better can presently be given than a heuristic analysis. This is not fully satisfactory, but it is better than having nothing at all. Such heuristic analyses add to our understanding of algorithms that are practically useful. They enable us to make comparisons between different algorithms, and to make predictions about their practical performance. If one insists on having fully proved theorems, then the best one can currently do is explicitly formulating all heuristic assumptions that enter into the analysis. For examples of such theorems we refer to [34]. For one factoring algorithm, the random class groups method, one of these heuristic assumptions turned out to be incorrect, and consequently the heuristic subexponential run time estimate for that algorithm had to be withdrawn (see [31]).

For the number field sieve the heuristic run time analysis is unusually laborious, and it is carried out in some detail in [7]. The algorithm in the present paper is sufficiently similar to that in [7] that we may content ourselves with indicating how the analysis in [7] needs to be modified, and what the outcome of the modified analysis is.

Our estimates will depend on $N = r^e - s$ rather than on the divisor n of N; in most cases N will be not much larger than n. We use the notation $L_x[v, \lambda]$ introduced in the Introduction. Also, the expression $L_N[v, \lambda + o(1)]$ will be abbreviated to $L_N[v, \lambda]$, here the $o(1)$ is for $N \to \infty$, uniformly for r, s in a finite set. We shall express our final estimates in the latter notation. We note that this makes sense only if r, s are fixed, or range over a finite set, and e tends to infinity.

6.1. *Probability of smoothness.* The result that makes the L-function useful in estimating smoothness probabilities reads as follows (cf. [25, (3.16); 7, Section 10]). Let $C \subset \mathbf{R}^4$ be a compact set such that for all $(\lambda, \mu, w, v) \in C$ one has $\lambda > 0$, $\mu > 0$, and $0 < w < v \leq 1$. Then the probability that a random positive integer $\leq L_x[v, \lambda]$ is $L_x[w, \mu]$-smooth equals $L_x[v - w, -\lambda(v - w)/\mu + o(1)]$ for $x \to \infty$, uniformly for (λ, μ, w, v) in C.

6.2. *Parameter choice as a function of the degree.* We begin by indicating the optimal choices of the parameters as a function of N and the degree d of the number field. These are derived from 6.1 by means of the heuristic argument that was presented in support of Conjecture 11.4 in [7]. The main change that

needs to be made is that the upper bound for $|(a+bm)\mathbf{N}(a+b\alpha)|$ used in [7] is replaced by the smaller value $(a_{\max}+b_{\max}m)(a_{\max}^d+b_{\max}^d|t|)$, where $m \approx N^{1/d}$ and where we assume $a_{\min} = -a_{\max}$. Following this change through the entire argument one finds that optimal choices of the parameters are obtained if all of $a_{\max}$, $b_{\max}$, B_1, and B_2 are taken equal to

$$\exp\Big(\big(\tfrac{1}{2}+o(1)\big)\Big(d\log d+\sqrt{(d\log d)^2+2\log(N^{1/d})\log\log(N^{1/d})}\Big)\Big),$$

the $o(1)$ being for $e \to \infty$, uniform for bounded r and s and for d in the region $1 < d^{2d^2} < N$. (The analysis in [7] assumes that $a_{\max} = b_{\max}$, $B_1 = B_2$, but this makes no difference.) In addition, we take $B_3 = B_1$ and $B_4 = B_2$, so that only full relations are considered (see 6.5). The size of the factor base and the number of full relations that one expects to find are given by the same expression. The typical size of the numbers $|(a+bm)\mathbf{N}(a+b\alpha)|$ that one wants to be smooth is

$$\exp\Big(\big(\tfrac{1}{2}+o(1)\big)\Big(d^2\log d+2\log(N^{1/d})+d\sqrt{(d\log d)^2+2\log(N^{1/d})\log\log(N^{1/d})}\Big)\Big).$$

The run time for the sieving in Step 2 and for the solution of the linear system in Step 3 each come out to be

$$\exp\Big((1+o(1))\Big(d\log d+\sqrt{(d\log d)^2+2\log(N^{1/d})\log\log(N^{1/d})}\Big)\Big).$$

The other parts of the algorithm take less time, with the possible exception of the search for G and U in Step 1, since this search has no equivalent in the algorithm of [7]; this point is discussed in 6.4.

6.3. *Optimal parameters.* The optimal choice for d as a function of N is given by

$$d = \left(\frac{(3+o(1))\log N}{2\log\log N}\right)^{1/3} \qquad \text{for } e \to \infty$$

uniformly for r, s in a finite set. With this choice for d, the choices for $a_{\max}$, $b_{\max}$, B_1, B_2, B_3, and B_4 made in 6.2 are $L_N[\frac{1}{3},(2/3)^{2/3}]$. The typical size of the numbers $|a+bm|$ and $|\mathbf{N}(a+b\alpha)|$ is $L_N[\frac{2}{3},(2/3)^{1/3}]$, so the numbers $|(a+bm)\mathbf{N}(a+b\alpha)|$ that one wants to be smooth are about $L_N[\frac{2}{3},(16/3)^{1/3}]$; this is $N^{o(1)}$, as announced in the introduction. The run time of the entire algorithm, with the possible exclusion of the search in the number field in Step 1, is $L_N[\frac{1}{3},(32/9)^{1/3}]$.

6.4. *Complexity of the search in the number field.* As we saw in Section 3, the search for U and G described in the first half of Section 3 is not likely to work in all cases. For this reason we consider instead the modifications described in 3.6 and 3.8. A routine calculation shows that the determination of G' in 3.6 can, for the parameter choices in 6.2 and 6.3, be performed within the same time limit. For the methods to determine U' that were indicated in 3.8 this is not

so clear. To illustrate the difficulty, let us consider the algorithm of Buchmann that was mentioned in 3.8. Its run time is, according to [30, Theorem 5; 6, Section 6], bounded by $(\log \Delta)^{cd} \cdot \sqrt{\Delta}$ for some absolute constant c. In our case we have $\Delta = d^{(1+o(1))d}$ for $e \to \infty$ (uniformly for bounded r, s), so that the run time estimate of Buchmann's algorithm becomes $d^{(c+\frac{1}{2}+o(1))d}$. Since the run time in 6.3 is $d^{(4+o(1))d}$, this leads to the question whether one can take $c \leq 3\frac{1}{2}$. We do not know the answer to this question, but we consider it likely that at least one of the methods suggested in 3.8 will run in time at most $d^{(4+o(1))d}$ with our choice of parameters. If this is the case, then the run time $L_N[\frac{1}{3}, (32/9)^{1/3}]$ mentioned in 6.3 indeed applies to the entire number field sieve. If it is not the case, we can still claim this run time for the algorithm of [7], when applied to integers of our special form. In the examples that we did, the search for U and G took only a very small fraction of the total run time.

It may also be possible to justify, along similar lines, the run time given in 6.2 as a function of d, though perhaps not for as wide a range of d as indicated in 6.2.

6.5. *Remark.* Because of our choice $B_3 = B_1$ and $B_4 = B_2$, partial relations and cycles among them were not considered in the version of the algorithm analyzed above. The use of partial relation is important for the practical performance of the number field sieve, as we shall illustrate in Section 8. Nevertheless, it is unlikely that the use of partial relations will affect the run time estimate by more than a factor $L_N[\frac{1}{3}, 0]$. For a run time analysis of the cycle finding algorithm and a discussion of the expected number of cycles we refer to [28].

7. Additional remarks

7.1. *Using more number fields.* Instead of using a single number field, as in 2.5, one can consider using several fields. Because the probability of finding relations decreases with growing b, this might be advantageous, because for each number field one can start afresh with the small b values. If we use smoothness bounds B_{1i} and B_{2i} for the ith number field K_i, then we need approximately $\max_i\{\pi(B_{1i})\} + \sum_i(\#U_i + \#G_i)$ relations, where U_i generates the units and G_i the first degree prime ideals of norm $\leq B_{2i}$ in K_i. Hence no K_i should be used that contributes fewer than $\#U_i + \#G_i$ relations.

To give an example of this multi-field approach, suppose that we want to factor an integer n of the form $2^{8e} + 1$, with e a positive integer, using number fields of degree 4. Direct application of the construction in 2.5 leads to the field $\mathbf{Q}(\zeta^2)$, where ζ is a primitive 16th root of unity; ζ^2 is a zero of the polynomial $X^4 + 1$, and it maps to $2^{2e} \bmod n$ under φ. Two other fourth degree fields that can be used are the fields $\mathbf{Q}(\zeta \pm \zeta^{-1})$, where $\zeta \pm \zeta^{-1}$ satisfies the polynomial $X^4 \mp 4X^2 + 2$ and is mapped to $(2^e \pm 2^{-e}) \bmod n = (2^e \mp 2^{7e}) \bmod n$ under φ. To the free relations from 2.13 one can then add the multiplicative relations that exist between elements of different fields K_i. As can be seen in [26], we did not use this approach for $e = 64$, and as far as we know the practical importance of the multi-field approach is still unexplored. From a theoretical point of view the idea has proved to be worthwhile, see [11].

7.2. *Postponing the construction of cycles.* The construction of cycles among the partial relations to obtain relations among the a_i can be postponed until Step 3, as mentioned in 2.12. Here we sketch how this can be achieved. Given a collection of partial relations, let P_1 and P_2, as in 2.12, be the sets of large primes and large prime ideals occurring in the partial relations, and let $\bar{I} = (I \backslash U) \cup P_1 \cup P_2$. Each partial relation can be regarded as an element $\bar{v} = (\bar{v}_i)_{i \in \bar{I}}$ of $\mathbf{Z}^{\bar{I}}$. For $i \in I \backslash U$ we have $\bar{v}_i = v_i$ as in Section 2, and for each $\bar{v}$ at least 1 and at most 2 of the $\bar{v}_i$ with $i \in P_1 \cup P_2$ are non-zero. Let $\bar{V}$ be the collection of $\bar{v}$'s, and let F be the collection of full relations that have been found.

Given $\bar{V}$, we attempt to find more than $\#I - \#F$ linearly independent linear combinations among its elements for which the entries corresponding to the $i \in P_1 \cup P_2$ are even. One way to do this is by means of the methods from [22; 37]. It is easy to see that such linear combinations correspond to cycles among the partial relations. With appropriate signs, they can be turned into relations among the a_i (as in (2.3)), where the unit contribution can be determined as before. Combined with the full relations this gives more than $\#I$ relations among the a_i, so that Step 3 can be completed in the usual manner.

There is no obvious way to see if a collection of partial relations will indeed give rise to more than $\#I - \#F$ linear combinations as above. In practice one could first use the cycle counter from [28], and only proceed with the matrix step above if there are enough cycles.

7.3. *Double large primes.* Following the approach from [28], we can allow two large primes exceeding B_1 in $a + bm$, or two large prime ideals of norms larger than B_2 in $a + b\alpha$. This variant of the algorithm turned out to be much slower than the version described in Section 2. This was caused by the dramatic growth of the number of reports and trial divisions in the sieving step. Most of these trial divisions were fruitless, partly because the large factors remaining after trial division were often found to be prime instead of the product of two large primes. It is possible, however, that this variant becomes preferable for larger values of n than we tried.

8. Examples

The first factorization obtained by means of the number field sieve was the factorization of the 39 digit number $F_7 = 2^{2^7} + 1$, which was in fact already known (see [32]). This factorization was carried out by the fourth author in 20 hours on a Philips P2012, an 8-bit computer with 64K of memory and two 640K disk drives. With $f = X^3 + 2$ and a factor base consisting of 500 rational primes, the units -1 and $1 + (-2)^{1/3}$, and 497 algebraic primes, it took 2000 values of b and per b the integers a with $|a| \leq 4800$ to find 538 *ff*'s and 1133 *pf*'s with $p_2 = 1$ and $B_3 = 10000$; no *fp*'s or *pp*'s were used. This led to 399 cycles, which combined with the 81 free relations (cf. 2.13) sufficed to factor F_7:

$$2^{128} + 1 = 59\,64958\,91274\,97217 \cdot 57\,04689\,20068\,51290\,54721.$$

Several steps of this first number field sieve factorization were not carried out as described in the previous sections. For instance, only the numbers $a + bm$

were sieved, prime powers were included in the sieving, and for the reports both $a + bm$ and $\mathbf{N}(a + b\alpha)$ were tested for smoothness by trial division. The unit contribution was found by means of a table containing u_1^i for $|i| \leq 8$. The fourth author was able to reduce the time needed for factoring F_7 by a factor of two by using some of the methods described in Sections 2, 4, and 5. Other numbers factored by the fourth author are $2^{144} - 3$ (44 digits, in 47 hours) and $2^{153} + 3$ (47 digits, in 61 hours):

$$2^{144} - 3 = 49\,27299\,91333 \cdot 45\,25956\,52604\,77899\,16201\,09802\,72761,$$
$$2^{153} + 3 = 5 \cdot 11 \cdot 6\,00696\,43200\,64900\,87537 \cdot 3455\,98297\,79603\,41893\,82757.$$

Other, and more general, factoring methods should actually be preferred for the factorization of integers in this range. We do not know for what size of numbers the number field sieve may be expected to be faster than other, asymptotically slower methods. We do know that for numbers of the right form that have more than 100 decimal digits the number field sieve is faster than the multiple polynomial quadratic sieve method. Until the appearance of the number field sieve, the quadratic sieve was the only algorithm by which numbers in the 100+ digit range without small factors could be factored, and it still has the advantage over the number field sieve that it applies to all numbers indiscriminately.

For our number field sieve implementation at Digital Equipment Corporation's Systems Research Center we followed the same approach as for our implementations of the elliptic curve method and quadratic sieve as described in [27]. In short, this means that one central processor distributes tasks among several hundred CVAX processors, the clients, and collects their results. For a more general set-up of the number field sieve, which also allows external sites to contribute to the factoring process by means of electronic mail, we refer to [26] and also [27]. This parallelization and distribution of tasks was used only for the second step of the algorithm, the collection of relations.

For the number field sieve tasks consist of short, non-overlapping intervals of b-values. When a client is given an interval $[b_1, b_1 + 1, \ldots, b_2]$, he starts sieving all pairs a, b for $b = b_1$, $b_1 + 1$, $\ldots$, b_2 in succession, and per b for $|a|$ less than some predetermined bound. After each b, the client reports the full and partial relations that it found for that b to the central processor (possibly no relations at all), and it reports that it just processed that particular value of b. The central processor keeps track of the relations it received and the b's that have been processed. It also notices if a client dies or becomes unavailable, which occurs for instance if a workstation is claimed by its owner. In that case the b's that are left unfinished by that client can be redistributed. In this way, all positive b's will be processed, without gaps, until sufficiently many relations have been collected.

This is a more conservative approach than we use for our elliptic curve and quadratic sieve implementations. For the latter algorithms we can afford not to worry about inputs that have been distributed but that are never processed. For the number field sieve the smaller b's are noticeably better than the larger ones, so that we decided to be careful and not to waste any of them.

TABLE 1. *Four factorizations obtained with the number field sieve*

$3^{239} - 1 = 2 \cdot 479 \cdot 17209 \cdot 4\,33019\,64055\,63553\,33339\,45745\,53310\,61280$
$44213 \cdot \text{p67};$

$2^{373} + 1 = 3 \cdot 60427 \cdot 694\,57949\,73168\,94264\,42566\,12436\,59806\,37197\,21883$
$18857 \cdot \text{p60};$

$7^{149} + 1 = 8 \cdot 10133 \cdot 4\,73384\,33355\,18992\,92791\,10650\,93183\,78061\,19829\,00857$
$39285\,01623 \cdot \text{p66};$

$2^{457} + 1 = 3 \cdot 6885\,35756\,02053\,19573\,06063\,38968\,00918\,44825\,49047\,29193 \cdot$
p89.

In Table 1 we list the first four factorizations that were obtained with our implementation of the number field sieve, with pi denoting a prime of i decimal digits. Additional data concerning these factorizations are found in Table 2. In the first case $\mathbf{Z}[\alpha]$ is a unique factorization domain. In the other three cases this is not true, but we could use instead the ring of integers of $\mathbf{Q}(\alpha)$, which does have unique factorization. This ring of integers is equal to $\mathbf{Z}[\alpha^3/2]$, $\mathbf{Z}[\alpha]+\mathbf{Z}\cdot(\alpha+2)^4/5$, and $\mathbf{Z}[\alpha^2/2]$ in the three respective cases.

Although the theoretical analysis indicates that the choice $B_1 = B_2$ is asymptotically optimal, one can imagine that in practice there are cases in which it is better to take B_1 much smaller or much larger than B_2. We have no experience with this. Introducing several fields as in 7.1 leads to an asymmetry between B_1 and B_2, see for example [11].

The first two factorizations could have been obtained with much smaller factor bases if we had used the *pp*'s, as we did for the other two. The first entry is the first number we collected relations for; even with our restricted use of the partial relations the factor base was chosen much too large. For the last two entries our choice of factor base size turned out to be much better. This was, in particular for the third entry, more or less a matter of luck, as we had no way to guess how many partial relations would be needed to produce a given number of cycles. The experience gained with these and other numbers (see [2; 26; 28]) enables us to select the bounds B_1 and B_2 in a slightly less uncertain manner.

Before one invests a lot of computing time in the search for relations, it is wise to check if the chosen values for B_1 and B_2 are likely to work. By processing several reasonably distanced intervals of consecutive b-values, one can get a fairly cheap and accurate estimate of the total yield of full and partial relations, which should help to decide if the choices are realistic. In our later experiments we tried to select B_1, B_2, and $b_{\max}$ such that the run time of Step 2 is minimized, and such that one quarter of the final set of relations consists of full relations and the remaining three quarters are expected to be produced by cycles among the partials. This is probably rather conservative, but given how the number of cycles varies, it seems to be a safe choice; in any case, we never had to start all over again with larger bounds.

For the factorizations reported in Table 1 the first step, the determination

TABLE 2. *Data on the four factorizations*

n is factor of	$3^{239}-1$	$2^{373}+1$	$7^{149}+1$	$2^{457}+1$
# digits of n	107	108	122	138
f	X^5-3	X^5+4	X^5+7	X^5+8
m	3^{48}	2^{75}	7^{30}	2^{92}
$B_1=B_2$	479910	287120	287120	479910
$\#P$	40000	25000	25000	40000
$\#U+\#G$	$3+40067$	$3+25010$	$3+24880$	$3+40012$
factor base size	80070	50013	49883	80015
$B_3=B_4$	10^8	10^8	10^8	10^8
$a_{\max}=-a_{\min}$	$5\cdot10^6$	$5\cdot10^6$	$5\cdot10^6$	$5\cdot10^6$
$b_{\max}$	120000	200000	1136000	2650000
# free's	2014	1248	1222	2003
# fulls	≈ 30000	≈ 20000	10688	17625
# partials	not kept	not kept	1358719	1741365
# *pf*, *pf* pairs	≈ 25000	≈ 15000	5341	not counted
# *fp*, *fp* pairs	≈ 25000	≈ 15000	5058	not counted
# cycles with *pp*'s	not used	not used	≈ 28000	not counted
total # cycles	>50000	>30000	≈ 38400	62842
run time Step 2	2 days	3 days	2 weeks	7 weeks
run time Step 3	2 weeks	4 days	5 days	2 weeks
# digits of factors	41 & 67	48 & 60	56 & 66	49 & 89

of sets of generators, turned out to be quite easy. In all four cases the set U consisted of -1 and two units of infinite order, which were not hard to come by. Determination of G by means of the method described in Section 3 never took more than fifteen minutes on a CVAX processor. In Step 2, we partitioned the a-interval into subintervals of length 500000.

One can find the cycles of length two in a trivial manner by sorting the *pf*'s (and *fp*'s) according to p_1 (and p_2), which is all we did to generate the cycles for the first two factorizations. For the third entry the yield was already getting quite low for b around 1100000, and we would never have been able to factor the third and the fourth number had we not used cycles involving *pp*'s as well.

To place the run times in perspective one should keep in mind that Step 2 was performed on a network of several hundred CVAX processors, whereas Step 3 was done on a single workstation containing six CVAX processors by means of a fairly elementary Gaussian elimination program. Since these factorizations were carried out we made substantial improvements in our implementation of the third step, see [26; 28]. Other numbers we factored are a composite 115 digit factor of $3^{241}-1$ into a p52 and a p57, a composite 108 digit factor of $6^{149}-1$ into p36 · p79, and a composite 117 digit factor of $3^{251}-1$ into p37 · p80. These factorizations did not produce new insights, and they were reported in the updates to [3]. Furthermore we factored the composite 148 digit factor of the ninth Fermat number, as reported in [26]. For more numbers we refer to [2].

9. Generalization

Following an idea of Buhler and Pomerance, we can attempt to generalize the number field sieve to integers n that do not have a small multiple of the form $r^e - s$, for small r and $|s|$, as follows. Select a positive integer d, an integer m that is a little smaller than $n^{1/d}$, and put $f = \sum_{i=0}^{d} f_i X^i$, where $n = \sum_{i=0}^{d} f_i m^i$ with $0 \leq f_i < m$. The algebraic number field is then defined as $K = \mathbf{Q}(\alpha)$ with $f(\alpha) = 0$, and the map $\varphi: \mathbf{Z}[\alpha] \to \mathbf{Z}/n\mathbf{Z}$ sends α to $(m \bmod n)$.

If one is very lucky one hits upon a value of m for which the resulting number field has a very small discriminant. This occurs, for example, if the digits f_i of n in base m are very small. In that case the algorithm as described in this paper can be applied without major changes. It is much more likely, however, that one is not so lucky, and then Steps 1 and 3 will run into serious trouble. It is debatable how probable it is that $\mathbf{Z}[\alpha]$ (or the ring of integers of K) is a unique factorization domain (see [9]); but even if it is, it is completely unrealistic to expect that the search methods discussed in Section 3 can be used to find generators for the unit group and for the first degree prime ideals. This is because the values for M and C would have to be taken prohibitively large. Standard estimates suggest that the coefficients of the elements of U and G, when written as explicit polynomials in α, are so large that they cannot even be written down in a reasonable amount of time, let alone calculated. This means that the elements of U and G must be represented in a different way, or that their computation must be avoided altogether. We discuss a variant of the number field sieve that accomplishes the latter.

9.1. *Elimination over* $\mathbf{Z}$. To describe this variant, we make the simplifying assumption that $\mathbf{Z}[\alpha]$ is the ring of integers of K; this assumption is discussed in 9.4. Also, we consider, for simplicity, only *full* relations. The sieving step provides us with many pairs of coprime integers a, b with the property that both $a + bm$ and $a + b\alpha$ are smooth:

$$a + bm = \prod_{p} p^{e_{a,b}(p)}, \qquad (a + b\alpha) = \prod_{\mathfrak{p}} \mathfrak{p}^{e_{a,b}(\mathfrak{p})}.$$

Here p ranges over the prime numbers $\leq B_1$, and $\mathfrak{p}$ over the first degree prime ideals of $\mathbf{Z}[\alpha]$ of norm $\leq B_2$; furthermore, the $e_{a,b}(p)$ and $e_{a,b}(\mathfrak{p})$ are non-negative integers that one can compute. Note that we use only the *ideal* factorization of $a + b\alpha$, not a factorization in terms of sets U and G. Next one looks for solutions to the system

$$\sum_{a,b} e_{a,b}(p) z_{a,b} \equiv 0 \bmod 2, \qquad \sum_{a,b} e_{a,b}(\mathfrak{p}) z_{a,b} = 0, \qquad z_{a,b} \in \mathbf{Z},$$

the sums ranging over the pairs a, b that have been found. This amounts to solving a large and sparse system of linear equations over $\mathbf{Z}$. If r_1, r_2 are as in Section 3, then one needs a little more than $r_1 + r_2$ solutions $z = (z_{a,b})$; they should be independent in a suitable sense. For each solution z, the integer

$$\prod_{a,b} (a + bm)^{z_{a,b}}$$

is the square of an integer that can be explicitly written down as a product of prime numbers $p \leq B_1$. Also, the exponents occurring in the prime ideal factorization of the algebraic number

$$u = \prod_{a,b} (a + b\alpha)^{z_{a,b}} \tag{9.2}$$

are all equal to 0, so u is a unit. Therefore each z gives rise to a relation of the type

$$\Big(\prod_p \varphi(p)^{w(p)}\Big)^2 = \varphi(u) \tag{9.3}$$

in $\mathbf{Z}/n\mathbf{Z}$, where u is a unit given by (9.2) and the $w(p)$ are integers. If one has sufficiently many such relations, then the units u become multiplicatively dependent, and one can find an explicit dependence relation by combining the techniques of Section 5 with lattice basis reduction. For this one needs to know the logarithms of the images of the units u under the embeddings $K \to \mathbf{C}$, and these can be computed from (9.2). Taking the corresponding product of the relations (9.3) one finds a relation of the type

$$\Big(\prod_p \varphi(p)^{y(p)}\Big)^2 = \varphi(1),$$

so that $x = \prod_p p^{y(p)}$ is a solution to (2.2), as required.

9.4. *The ring of integers.* In 9.1 we made the assumption that $\mathbf{Z}[\alpha]$ is the ring of integers of K. If this condition is not satisfied then some of the elements u produced by the algorithm may not be units. In that case the vectors $\nu(u)$ from Section 5 will not necessarily belong to a lattice, so that lattice basis reduction techniques cannot be used to find relations with integer coefficients between these vectors.

There are several ways to deal with this problem. The first is based on the following conjecture, which is quite possibly provable with present-day techniques: let d be an integer, $d \geq 2$; then for a "random" polynomial $f = \sum_{i=0}^{d} f_i X^i$ with $f_i \in \mathbf{Z}$, $f_d = 1$, the condition that $\mathbf{Z}[\alpha]$ is the ring of integers of K is satisfied with probability equal to $6/\pi^2$. This conjecture suggests that when one tries a few values for m one soon runs into one for which the condition is satisfied, so that the algorithm does not encounter the difficulty just indicated.

Alternatively, one might deal with the problem by replacing $\mathbf{Z}[\alpha]$ by the ring of integers A of K. With from some minor adjustments to the algorithm (see 3.5) this ring can take the place of $\mathbf{Z}[\alpha]$. One might object that the only known algorithms for determining A that are sufficiently fast for our purpose may fail (see [30, Section 4; 5]); more precisely, they do produce a subring A' of A, but A' may be different from A if the discriminant of f has a very large but unknown repeated prime factor. Fortunately, one can prove that A' works just as well as A if any such prime factor exceeds B_2.

9.5. *Complexity.* In the rest of this section we abbreviate $L_n[v, \lambda + o(1)]$, for $n \to \infty$, to $L_n[v, \lambda]$. Arguments similar to those in Section 6 suggest that the variant of the number field sieve that we just discussed factors any integer n in time $L_n[\frac{1}{3}, 9^{1/3}]$, where $9^{1/3} \doteq 2.0801$. A bottleneck is formed by the large linear system, which is to be solved over $\mathbf{Z}$ rather than over $\mathbf{Z}/2\mathbf{Z}$. We did invent a fairly complicated technique by which we could reduce the size of this system to approximately its square root, while preserving the sparsity; this technique depends on the availability—from a suitably modified Step 2—of many pairs of coprime integers a, b for which $a + b\alpha$ is B_2-smooth (with no condition on $a + bm$). This reduces the conjectural run time of the number field sieve to $L_n[\frac{1}{3}, (64/9)^{1/3}]$, where $(64/9)^{1/3} \doteq 1.9230$.

9.6. *Quadratic characters.* Although the ideas exposed above may have some use in practice, our discussion has been rather sketchy. This is because there exists a method that achieves the same conjectural run time $L_n[\frac{1}{3}, (64/9)^{1/3}]$ in a conceptually much simpler way. It employs quadratic characters in the number field. They were suggested by Adleman [1] as a tool to avoid both the assumption that the ring of integers of K is a unique factorization domain and the determination of the sets U and G. In [7] it was shown that quadratic characters can also be used to avoid the need to determine the ring of integers of K (cf. 9.4), a problem that is not dealt with in [1]. For a description of the method we refer to [1; 7].

It is an essential feature of the use of quadratic characters that it produces a square in the number field without producing its square root. This leads to the problem of computing square roots of very large algebraic integers. The known methods for doing this, which are discussed on [7, Section 9], lead to arithmetic operations with integers whose number of bits is roughly proportional to the square root of the run time of the entire factoring algorithm! A method proposed by Couveignes (see [15; 2]) works with much smaller numbers; it works only if the degree d is odd.

It is as yet unknown which of 9.1 and 9.6 should be preferred in practice. The first method has the disadvantage of a considerably more complicated elimination step, the second method requires substantial computations in the number field, but, as shown in [2], works quite satisfactorily if the extension degree is odd.

Coppersmith [11] showed that one can reduce the conjectural run time to $L_n[\frac{1}{3}, c]$, where $c = \frac{1}{3}(92 + 26\sqrt{13})^{1/3} \doteq 1.9019$, by using several number fields. There is no indication that the modification proposed by Coppersmith has any practical value.

REFERENCES

1. L.M. Adleman, *Factoring numbers using singular integers*, Proc. 23rd Annual ACM Symp. on Theory of Computing (STOC), New Orleans, May 6–8, 1991, 64–71.
2. D.J. Bernstein, A.K. Lenstra, *A general number field sieve implementation*, this volume, pp. 103–126.
3. J. Brillhart, D.H. Lehmer, J.L. Selfridge, B. Tuckerman, S.S. Wagstaff, Jr., *Factorizations of $b^n \pm 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$ up to high powers*, second edition, Contemp. Math.

22, Amer. Math. Soc., Providence, 1988.

4. J. Buchmann, *Complexity of algorithms in algebraic number theory*, in: R. A. Mollin (ed.), *Proceedings of the first conference of the Canadian Number Theory Association*, De Gruyter, Berlin, 1990, 37–53.
5. J. Buchmann, H. W. Lenstra, Jr., *Approximating rings of integers in number fields*, in preparation.
6. J. Buchmann, V. Shoup, *Constructing nonresidues in finite fields and the extended Riemann hypothesis*, in preparation. Extended abstract: Proc. 23rd Annual ACM Symp. on Theory of Computing (STOC), New Orleans, May 6–8, 1991, 72–79.
7. J. P. Buhler, H. W. Lenstra, Jr., C. Pomerance, *Factoring integers with the number field sieve*, this volume, pp. 50–94.
8. H. Cohen, *A course in computational algebraic number theory*, Springer-Verlag, to appear.
9. H. Cohen, H. W. Lenstra, Jr., *Heuristics on class groups*, pp. 33–62 in: H. Jager (ed.), *Number theory, Noordwijkerhout* 1983, Lecture Notes in Math. **1068**, Springer-Verlag, Heidelberg.
10. D. Coppersmith, *Fast evaluations of logarithms in fields of characteristic* 2, IEEE Trans. Inform. Theory **30** (1984), 587–594.
11. D. Coppersmith, *Modifications to the number field sieve*, J. Cryptology, to appear; IBM Research Report RC 16264, 1990.
12. D. Coppersmith, *Solving linear equations over $GF(2)$: block Lanczos algorithm*, Linear Algebra Appl., to appear; IBM Research Report RC 16997, 1991.
13. D. Coppersmith, *Solving linear equations over $GF(2)$ II: block Wiedemann algorithm*, Math. Comp., to appear; IBM Research Report RC 17293, 1991.
14. D. Coppersmith, A. M. Odlyzko, R. Schroeppel, *Discrete logarithms in $GF(p)$*, Algorithmica **1** (1986), 1–15.
15. J.-M. Couveignes, *Computing a square root for the number field sieve*, this volume, pp. 95–102.
16. J. D. Dixon, *Asymptotically fast factorization of integers*, Math. Comp. **36** (1981), 255–260.
17. T. ElGamal, *A subexponential-time algorithm for computing discrete logarithms over $GF(p^2)$*, IEEE Trans. Inform. Theory **31** (1985), 473–481.
18. D. M. Gordon, *Discrete logarithms in $GF(p)$ using the number field sieve*, SIAM J. Discrete Math. **6** (1993), 124–138.
19. D. M. Gordon, K. S. McCurley, *Massively parallel computation of discrete logarithms*, Advances in cryptology, Crypto '92, to appear.
20. J. L. Hafner, K. S. McCurley, *Asymptotically fast triangularization of matrices over rings*, SIAM J. Comput. **20** (1991), 1068–1083.
21. D. E. Knuth, *The art of computer programming*, volume 2, *Seminumerical algorithms*, second edition, Addison-Wesley, Reading, Massachusetts, 1981.
22. B. A. LaMacchia, A. M. Odlyzko, *Solving large sparse systems over finite fields*, Advances in cryptology, Crypto '90, Lecture Notes in Comput. Sci. **537** (1991), 99–129.
23. B. A. LaMacchia, A. M. Odlyzko, *Computation of discrete logarithms in prime fields*, Designs, Codes and Cryptography **1** (1991), 47–62.
24. S. Lang, *Algebra*, third edition, Addison-Wesley, Reading, Massachusetts, 1993.
25. A. K. Lenstra, H. W. Lenstra, Jr., *Algorithms in number theory*, Chapter 12 in: J. van Leeuwen (ed.), *Handbook of theoretical computer science*, Volume A, *Algorithms and complexity*, Elsevier, Amsterdam, 1990.
26. A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, J. M. Pollard, *The factorization of the ninth Fermat number*, Math. Comp. **61** (1993), to appear.
27. A. K. Lenstra, M. S. Manasse, *Factoring by electronic mail*, Advances in cryptology, Eurocrypt '89, Lecture Notes in Comput. Sci. **434** (1990), 355–371.
28. A. K. Lenstra, M. S. Manasse, *Factoring with two large primes*, Math. Comp., to appear.
29. H. W. Lenstra, Jr., *Factoring integers with elliptic curves*, Ann. of Math. **126** (1987), 649–673.

30. H. W. Lenstra, Jr., *Algorithms in algebraic number theory*, Bull. Amer. Math. Soc. **26** (1992), 211–244.
31. H. W. Lenstra, Jr., C. Pomerance, *A rigorous time bound for factoring integers*, J. Amer. Math. Soc. **5** (1992), 483–516.
32. M. A. Morrison, J. Brillhart, *A method of factoring and the factorization of* F_7, Math. Comp. **29** (1975), 183–205.
33. M. Pohst, H. Zassenhaus, *Algorithmic algebraic number theory*, Cambridge University Press, Cambridge, 1989.
34. C. Pomerance, *Analysis and comparison of some integer factoring algorithms*, pp. 89–139 in: H. W. Lenstra, Jr., R. Tijdeman (eds), *Computational methods in number theory*, Math. Centre Tracts **154/155**, Mathematisch Centrum, Amsterdam, 1983.
35. C. Pomerance, *Fast, rigorous factorization and discrete logarithm algorithms*, in: D. S. Johnson, T. Nishizeki, A. Nozaki, H. S. Wilf (eds), *Discrete algorithms and complexity*, Academic Press, Orlando, 1987, 119–143.
36. C. Pomerance (ed.), *Cryptology and computational number theory*, Proc. Sympos. Appl. Math. **42**, Amer. Math. Soc., Providence, 1990.
37. C. Pomerance, J. W. Smith, *Reduction of huge, sparse matrices over finite fields via created catastrophes*, Experiment. Math. **1** (1992), 89–94.
38. C. P. Schnorr, *Refined analysis and improvements on some factoring algorithms*, J. Algorithms **3** (1982), 101–127.
39. O. Schirokauer, *On pro-finite groups and on discrete logarithms*, Ph. D. thesis, University of California, Berkeley, 68 pages, May 1992.
40. I. N. Stewart, D. O. Tall, *Algebraic number theory*, second edition, Chapman and Hall, London, 1987.
41. B. Vallée, *Generation of elements with small modular squares and provably fast integer factoring algorithms*, Math. Comp. **56** (1991), 823–849.
42. D. Wiedemann, *Solving sparse linear equations over finite fields*, IEEE Trans. Inform. Theory **32** (1986), 54–62.
43. H. G. Zimmer, *Computational problems, methods and results in algebraic number theory*, Lecture Notes in Math. **262**, Springer-Verlag, Berlin, 1972.

ROOM MRE-2Q334, BELLCORE, 445 SOUTH STREET, MORRISTOWN, NJ 07960, U.S.A.
E-mail address: lenstra@bellcore.com

DEPARTMENT OF MATHEMATICS, UNIVERSITY OF CALIFORNIA, BERKELEY, CA 94720, U.S.A.
E-mail address: hwl@math.berkeley.edu

DEC SRC, 130 LYTTON AVENUE, PALO ALTO, CA 94301, U.S.A.
E-mail address: msm@src.dec.com

TIDMARSH COTTAGE, MANOR FARM LANE, TIDMARSH, READING, BERKSHIRE, RG8 8EX, ENGLAND