The number field sieve

A.K. Lenstra
Bellcore, 435 South Street, Morristown, NJ 07960
H.W. Lenstra, Jr.
Department of Mathematics, University of California, Berkeley, CA 94720
M.S. Manasse
DEC SRC, 130 Lytton Avenue, Palo Alto, CA 94301
J.M. Pollard
Tidmarsh Cottage, Manor Farm Lane, Tidmarsh, Reading, Berkshire, RG8 8EX, United Kingdom

Abstract. The number field sieve is an algorithm to factor integers of the form $r^e \pm s$ for small positive r and s. This note is intended as a 'report on work in progress' on this algorithm. We informally describe the algorithm, discuss several implementation related aspects, and present some of the factorizations obtained so far.

We also mention some solutions to the problems encountered when generalizing the algorithm to general integers using an idea of Buhler and Pomerance. It is not unlikely that this leads to a general purpose factoring algorithm that is asymptotically substantially faster than the fastest factoring algorithms known so far, like the multiple polynomial quadratic sieve.

1. Introduction

Since the introduction of the elliptic curve factoring algorithm in 1985 we have not seen any significant theoretical advances in integer factoring algorithms. Existing algorithms, like the multiple polynomial quadratic sieve algorithm (mpqs, the fastest practical general purpose factoring algorithm), have been polished both theoretically and practically. Although these efforts have pushed our factorization capabilities from the eighty digit range, through the nineties, to integers having more than one hundred digits [1, 4, 10, 14], cryptographers still feel confident basing the security of some of their cryptosystems on the supposed intractability of the factoring problem.

It is unlikely that the method presented here will have a major impact on the security of cryptosystems. However, it does make the integer factoring problem less intractable than many people expected it to be. We will present a special purpose factoring algorithm, the *number field sieve*, that is asymptotically faster than any other algorithm we know of, for the class of numbers it applies to. The algorithm has proved to be quite practical: it took us only a few weeks to factor numbers that would have taken several years had we used mpqs. Several aspects of our implementation will be discussed.

It seems that a suitable version of the number field sieve factors an integer n of the form $r^e \pm s$ in expected time

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

with $c = 2(2/3)^{2/3} \simeq 1.526$, irrespectively of the size of the factors of n, for r and $|s|$ below a fixed upper bound. This is substantially better than mpqs, which runs in heuristic expected time

(1.2) $\exp((1+o(1))(\log n)^{1/2}(\log\log n)^{1/2})$,

also independently of the size of the factors of n. Other factoring algorithms achieve the same running time as mpqs, heuristically or rigorously, but generally they are less practical than mpqs. Some people suspected that the running time in (1.2) would be the best we could ever achieve for factoring.

Unfortunately we are unable to give a rigorous proof that (1.1) is indeed the expected running time of the number field sieve. Consequently, this paper does not contain a rigorous mathematical result. In this context the following quotation from Donald Knuth (cf. [6]) is of interest: 'One of my mathematician friends told me he would be willing to recognize computer science as a worthwhile field of study, as soon as it contains 1000 deep theorems. This criterion should obviously be changed to include algorithms as well as theorems, say 500 deep theorems and 500 deep algorithms.' The present paper describes a deep algorithm for the solution of a fundamental problem, and it depends on techniques that have not been of traditional use in this area. We therefore trust that it is of interest to theoretical computer scientists, and that they will appreciate the challenge posed by its rigorous running time analysis. For a non-rigorous analysis, and a further discussion of heuristic estimates in integer factorization algorithms, we refer to Section 3.

As Joe Buhler and Carl Pomerance observed, the idea of the number field sieve can be applied to general integers as well. We present some suggestions how this generalization can be made to work in theory. It is suspected that the resulting algorithm runs in the same heuristic expected time (1.1), with $c = 3^{2/3} \approx 2.080$. If this turns out to be the case we would finally have an algorithm that is faster than (1.2), thereby settling the question about the optimality of (1.2). The practical consequences of this new general purpose factoring algorithm remain to be seen.

2. The algorithm

Let n be a composite integer of the form $r^e - s$, for a small positive integer r and a non-zero integer s of small absolute value. Examples of such n can be found in the Cunningham tables [2]. We describe a factoring algorithm, the number field sieve, that makes use of the special form of n. If n does not have the proper form, but a small multiple of n does, as is often the case on the 'wanted' lists from [2], the algorithm can be applied to this multiple of n.

The algorithm makes use of some elementary algebraic number theory. For background on this subject we refer to [15]. Given $n = r^e - s$, we first select an extension degree $d \in \mathbf{Z}_{>0}$. Given d, let $k \in \mathbf{Z}_{>0}$ be minimal such that $kd \geq e$, so that $r^{kd} \equiv sr^{kd-e} \bmod n$. Put $m = r^k$ and $c = sr^{kd-e}$, so that $m^d \equiv c \bmod n$, and let $f(X) = X^d - c \in \mathbf{Z}[X]$. For a reasonable choice of d a non-trivial factor of f will lead to a non-trivial factor of n, so that we may assume that f is irreducible. This enables us to define our number field K as $\mathbf{Q}(\alpha)$, where α satisfies $f(\alpha) = 0$. By ϕ we will denote the ring homomorphism from $\mathbf{Z}[\alpha]$ to $\mathbf{Z}/n\mathbf{Z}$ that sends α to $m \bmod n$.

The irreducibility of f can easily be checked: f is reducible if and only if either there is a prime p dividing d such that c is a pth power, or 4 divides d and $-4c$ is a fourth power (cf. [8, Ch. VIII, Thm. 9.1]). So, for example, if $s = 1$ we must have $\gcd(d, e) = 1$, but $\gcd(d, e)$ may be a power of 2 if $s = -1$.

Although it will not be the case in general, we will assume that $\mathbf{Z}[\alpha]$ is a unique factorization domain. This implies that the ring of integers of K equals $\mathbf{Z}[\alpha]$, so that we do not have to worry about denominators in the description of the algorithm. The algorithm can be made to work in the general case as well. To give an example, for $3^{239}-1$ (one of the numbers we factored, cf. Section 6), we used the number field $\mathbf{Q}(3^{1/5})$, so $d = 5$, $m = 3^{48}$, and $f(X) = X^5 - 3$; the ring $\mathbf{Z}[3^{1/5}]$ is indeed a unique factorization domain.

The idea of the number field sieve is to look for pairs of small coprime integers a and b such that both the algebraic integer $a + \alpha b$ and the integer $a + mb$ are smooth. Because $\phi(a + \alpha b) = (a + mb \bmod n)$, each pair provides a congruence modulo n between two products. Sufficiently many of those congruences can then be used to find solutions to $y^2 \equiv z^2 \bmod n$, which in turn might lead to a factorization of n. This method evolved from a method based on Gaussian integers from [5].

An algebraic integer is smooth if it can only be divided by prime ideals of $\mathbf{Z}[\alpha]$ of small norm. We can restrict ourselves to the prime ideals in $\mathbf{Z}[\alpha]$ of prime norm, since those are the only ones that can contain algebraic integers of the form $a + \alpha b$ with a and b coprime. The set of prime ideals of $\mathbf{Z}[\alpha]$ of prime norms is in 1-1 correspondence with the set of pairs p, c_p, where p is a prime number and $c_p \in \{0, 1, ..., p-1\}$ satisfies $f(c_p) \equiv 0 \bmod p$: for each pair p, c_p a prime ideal of norm p is generated by p and $\alpha - c_p$, or equivalently, the prime ideal is the kernel of the ring homomorphism from $\mathbf{Z}[\alpha]$ to $\mathbf{Z}/p\mathbf{Z}$ that sends α to c_p. In particular, $a + \alpha b$ is in the prime ideal corresponding to p, c_p if and only if $a + c_p b \equiv 0 \bmod p$.

The prime ideal factorization of $a + \alpha b$ corresponds to the factorization of the norm $N(a + \alpha b) = a^d - c(-b)^d \in \mathbf{Z}$ of $a + \alpha b$, if a and b are coprime: if $a^d - c(-b)^d$ has exactly k factors p, with $k > 0$, then $a \equiv -c_p b \bmod p$ for a unique root c_p of f modulo p, and the prime ideal corresponding to the pair p, c_p divides $a + \alpha b$ exactly to the kth power. So, one ideal of norm p takes care of the full exponent of p in $a^d - c(-b)^d$.

We give a more precise description of the algorithm. After selecting K, we first fix a smoothness bound $B \in \mathbf{R}_{>0}$. The value for B is best determined experimentally. Let a and b be integers with $b \geq 1$ and $\gcd(a, b) = 1$. Suppose that both $N(a + \alpha b)$ and $a + mb$ are B-smooth, i.e.,

$$(2.1) \quad |N(a + \alpha b)| = \prod_{p \text{ prime}, p \leq B} p^{v_p},$$

and

$$(2.2) \quad |a + mb| = \prod_{p \text{ prime}, p \leq B} p^{w_p},$$

for $v_p, w_p \in \mathbf{Z}_{\geq 0}$. Suppose furthermore that we can use the prime factorization of $N(a + \alpha b)$ to derive a factorization of $a + \alpha b$ into units and prime elements, i.e.,

$$(2.3) \quad a + \alpha b = \left(\prod_{u \in U} u^{t_u}\right) \cdot \left(\prod_{g \in G} g^{v_g}\right),$$

for $t_u \in \mathbf{Z}$ and $v_g \in \mathbf{Z}_{\geq 0}$. Here U is some predetermined set of generators of the group of units and G is a set of generators of the prime ideals of $\mathbf{Z}[\alpha]$ of prime norms $\leq B$. It follows that

$$(2.4) \quad \left(\prod_{u \in U} \phi(u)^{t_u}\right) \cdot \left(\prod_{g \in G} \phi(g)^{v_g}\right) = \left(\prod_{p \text{ prime}, p \leq B} p^{w_p} \bmod n\right),$$

where a minor change of the t_u's, if necessary, takes care of the sign of $a + mb$.

If we have more than $\#U + \#G + \pi(B)$ (with $\pi(B)$ the number of primes $\leq B$) of such pairs a, b, then we can find integers $x(a, b) \in \{0, 1\}$, not all zero, such that at

the same time

$$(2.5)\quad \prod_{a,b}(a+\alpha b)^{x(a,b)} = \Big(\Big(\prod_{u\in U} u^{\bar{l}_u}\Big)\cdot\Big(\prod_{g\in G} g^{\bar{v}_g}\Big)\Big)2$$

and

$$(2.6)\quad \prod_{a,b}(a+mb)^{x(a,b)} = \Big(\prod_{p \text{ prime}, p\le B} p^{\bar{w}_p}\Big)2,$$

so that,

$$(2.7)\quad \Big(\Big(\prod_{u\in U}\phi(u)^{\bar{l}_u}\Big)\cdot\Big(\prod_{g\in G}\phi(g)^{\bar{v}_g}\Big)\Big)2 = \Big(\Big(\prod_{p \text{ prime}, p\le B} p^{\bar{w}_p}\Big)2 \bmod n\Big),$$

where $\bar{l}_u \in \mathbf{Z}$, and $\bar{v}_g, \bar{w}_p \in \mathbf{Z}_{\ge 0}$. Such $x(a,b)$ can for instance be found by applying Gaussian elimination modulo 2 to the vectors consisting of the exponents in (2.4). From (2.7) we find integers y and z with $y^2 \equiv z^2 \bmod n$. A possibly non-trivial factorization of n then follows by computing $\gcd(n, y-z)$. It should be noted that each new solution $x(a,b)$ to (2.5) and (2.6) gives a new pair y, z, and thus another chance of factoring n.

To turn the above description into an algorithm, we have to answer the following questions:

1 - Given B, how do we find 'good' pairs a, b, i.e., pairs a, b such that both (2.1) and (2.2) hold?

2 - How do we find sufficiently many good pairs?

3 - How do we find a set U of generators of the group of units?

4 - How do we find a set G of generators of the prime ideals of $\mathbf{Z}[\alpha]$ of prime norms $\le B$?

5 - How do we turn (2.1) into (2.3)?

6 - What is the expected running time of the resulting algorithm?

And, less important for the moment, but of considerable practical interest:

7 - Can we take advantage of large primes in (2.1) and (2.2)?

The remainder of this section will be devoted to Questions 1 through 5. Questions 6 and 7 will be discussed in Sections 3 and 4, respectively.

Finding good pairs.

Concerning question 1 we note in the first place that for each fixed b the $a+mb$'s can be tested for B-smoothness using a sieve over some suitable interval of a-values. For a prime p the starting point for the sieve equals $-mb \bmod p$, so that the starting point for the sieve for $b+1$ follows by subtracting $m \bmod p$ from the starting point for b. Sequences of consecutive b-values can therefore be processed quite efficiently (i.e., without divisions) once $m \bmod p$ has been computed for all primes $p \le B$, and the computation has been set up for some initial b.

For pairs a, b for which $a+mb$ is B-smooth we could test $N(a+\alpha b) = a^d - c(-b)^d$ for smoothness by trial division. If there are only a few smooth $a+mb$'s per b this might be a reasonable idea. Usually however, there will be far too many smooth $a+mb$'s per b (only a few of which, if any, will lead to a smooth $N(a+\alpha b)$) to make this efficient. We found that it is far more efficient to test the norms for smoothness using another sieve.

This can easily be achieved if we use the pairs p, c_p as defined above, because a prime p divides $a^d - c(-b)^d$ if and only if $a \equiv -c_p b \bmod p$ for some c_p. So, to be able to sieve efficiently, it suffices to compute all pairs p, c_p with $f(c_p) \equiv 0 \bmod p$ for the primes $p \le B$. The number of pairs we get in this way equals $\#G$ and will turn out to be approximately equal to the number of primes $\le B$. The pairs p, c_p should be computed once, using for instance a probabilistic polynomial root finder over finite fields (cf. [7]), and stored in a file.

For each b and some interval of a-values we generated the good pairs a, b as follows:

- Initialize all sieve locations to zero.
- Sieve the $a+mb$'s by adding $[\log_2 p]$ to the appropriate sieve locations for the primes $p \le B$. The starting points can usually be found either by adapting information from the previous b, or by using the end points from the previous interval of a's. Small primes can be replaced by their powers to make this step go faster.
- Check the sieve locations. For a location that contains a value close to $\log_2(a+mb)$ and for which $\gcd(a, b) = 1$, replace that sieve location by zero. Otherwise, replace that sieve location by a sufficiently small negative number.
- Sieve the $a^d - c(-b)^d$'s by adding $[\log_2 p]$ to the appropriate sieve locations for all pairs p, c_p with $p \le B$. The starting points are again easy to compute, once the computation has been set up.
- Check the sieve locations. For a location that contains a value close to $\log_2(a^d - c(-b)^d)$, attempt to factor $a+mb$ by trial division using the primes $\le B$. If the factorization attempt is successful, attempt to factor $a^d - c(-b)^d$ by trial division using the primes $\le B$. If the factorization attempt is successful, a good pair a, b has been found.

Notice that for each b and interval of a-values we sieve twice but use the same memory locations for the sieve locations. We found that this is faster than sieving with $a+mb$ and $a^d - c(-b)^d$ at the same time, again using one memory location per sieve location, because of the large number of false reports we got in that case. This latter problem can be avoided by using two memory locations per sieve location.

Finding sufficiently many pairs.

To answer the second question, just apply the above to $b = 1, 2, \ldots$ in succession, until sufficiently many good

pairs have been found. There is one problem, however. The bigger b gets, the smaller the probability that both $a+mb$ and $a^d-c(-b)^d$ are B-smooth. In practice this means that the yield becomes quite noticeably lower and lower, and that it might be impossible ever to find sufficiently many good pairs. For the moment there is not much we can do about this (but see also Section 4). The only remedy seems to be to select a bigger B, and try again.

Finding U.

Let r_1 be the number of real roots of f, and $l = ((d+r_1)/2)-1$. Notice that for our type of polynomial r_1 will be 0, 1, or 2. The group of units is generated by an appropriate root of unity u_0 and l independent elements, say $u_1, u_2, ..., u_l$, of infinite order. Notice that $u_0 = -1$ if $r_1 > 0$.

Compute the norms of many elements of the form $\sum_{i=0}^{d-1} a_i \alpha^i \in \mathbf{Z}[\alpha]$ for a_i's with small absolute value. In that way it is usually not hard to find l multiplicatively independent elements $u_1, u_2, ..., u_l$ with norm equal to ± 1 that together with u_0 generate the group of units. Later we will see that, if $r_1 > 0$, it is useful to require that some particular real embedding of the u_i's is positive. If $r_1 > 0$, we fix one particular real embedding for this purpose. We put $U = \{u_0, u_1, ..., u_l\}$; if necessary we later change the set U as explained below ('Finding the unit contribution'). Finding U can also be done while finding G.

Finding G.

Finding a set G of generators of the prime ideals of $\mathbf{Z}[\alpha]$ of prime norms $\leq B$ is more challenging, but can be done in more or less the same way. As we have seen above, the $\#G$ pairs p, c_p with $p \leq B$ are in 1-1 correspondence with the prime ideals of $\mathbf{Z}[\alpha]$ of prime norms $\leq B$. So, for each of the $\#G$ pairs p, c_p it is our task to find a generator $g(p, c_p)$ of the prime ideal corresponding to p, c_p, i.e., an expression of the form $\sum_{i=0}^{d-1} g_i \alpha^i \in \mathbf{Z}[\alpha]$, with $g_i \in \mathbf{Z}$, of norm equal to $\pm p$ for which $\sum_{i=0}^{d-1} g_i c_p^i \equiv 0 \bmod p$. If $r_1 > 0$ we require that the real embedding, as fixed above, of $g(p, c_p)$ is positive.

Let a_B and C_B be two positive constants depending on B (and on K). First, put $a(p, c_p) = a_B$ for all pairs on the list. Next, for all $h = \sum_{i=0}^{d-1} h_i \alpha^i \in \mathbf{Z}[\alpha]$, with $h_i \in \mathbf{Z}$, for which $\sum_{i=0}^{d-1} h_i^2 |\alpha|^{2i} \leq C_B$ and $N(h)$ is of the form kp for some prime p from our list of pairs and some non-zero integer k with $|k| < \min(p, a_B)$, do the following. Find the root c_p corresponding to p and h, i.e., c_p such that $\sum_{i=0}^{d-1} h_i c_p^i \equiv 0 \bmod p$; if $|a(p, c_p)| > |k|$ then replace $a(p, c_p)$ by k and put $\bar{g}(p, c_p)$ equal to h.

If a_B and C_B were chosen properly, we should now have that $a(p, c_p) < a_B$ for all pairs p, c_p. In practice most $a(p, c_p)$ will be equal to ± 1; for those pairs we have that $g(p, c_p) = \bar{g}(p, c_p)$. For the pairs for which $a(p, c_p) \neq \pm 1$, we compute $g(p, c_p)$ by dividing $\bar{g}(p, c_p)$ by the appropriate generator of an ideal of norm $a(p, c_p)$; this will require the computation of only a very limited number of inverses of elements of K. If $r_1 > 0$, replace $g(p, c_p)$ by $-g(p, c_p)$, if necessary, to make its real embedding positive; if d is odd we can look at the sign of $N(g(p, c_p))$ instead. In the full version of this paper it will be explained how a_B and C_B should be chosen; in practice it suffices to try several values until it works.

Finding the unit contribution.

Now how do we use our good pairs, and our sets U and G to produce relations that are useful for factoring n, i.e., how do we turn (2.1) into (2.3)? Above we saw how the factorization of $a^d-c(-b)^d$ can be used to obtain the factorization of $a+\alpha b$ as a product of powers of prime ideals. Replacing, in this product, the prime ideals by their generators, we find an element that multiplied by a suitable unit equals $a+\alpha b$. The problem is to find this unit, and to express it as a product of elements of U.

In principle one can find the unit by performing divisions in the number field, and then express it in U by table look-up. A faster method keeps track of some extra information per generator, and uses vector additions instead of polynomial multiplications modulo f. Let $U = \{u_0, u_1, ..., u_l\}$ be as constructed above. Choose l embeddings of K into $\mathbf{C}$ such that no two are complex conjugates, and denote, for $x \in K$, by x_i the image of x under the ith embedding, for $i = 1, 2, ..., l$. Let

$$v(x) = (\log|x_1| - (\log|N(x)|)/d, \log|x_2| - (\log|N(x)|)/d, ..., \log|x_l| - (\log|N(x)|)/d) \in \mathbf{R}^l,$$

for $x \in K$. Under the mapping v the group of units forms a lattice in $\mathbf{R}^l$. Let W be the $l \times l$ matrix having $v(u_i)^T$ as ith column, for $i = 1, 2, ..., l$. Then the columns of W form a basis for this lattice.

Now to write $(a+\alpha b)/\prod g(p, c_p)^{v_g}$ as a product of elements of U, simply compute

$$W^{-1} \cdot (v(a+\alpha b) - \sum v_g \cdot v(g(p, c_p)));$$

the ith element of this integral vector equals the number of times u_i occurs in the quotient, for $i = 1, 2, ..., l$. If $r_1 > 0$, the u_0 contribution will be equal to the sign of the real embedding of $a+\alpha b$, if the u_i and the $g(p, c_p)$ have been chosen such that their real embeddings are positive (where we use the real embedding that has been fixed above). If $r_1 = 0$, the u_0 contribution can be found by keeping track of the arguments of a particular complex embedding of the u_i and the $g(p, c_p)$.

In practice the mapping v and the entries of W^{-1} will only be computed in limited precision, and the entries of the above vector will have to be 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 $g(p, c_p)$ such that the coordinates of $v(g(p, c_p))$ lie between $-1/2$ and $1/2$; this can be achieved by multiplying $g(p, c_p)$ by some appropriate product of units (to be determined using v). Notice that the resulting $v(g(p, c_p))$ need be computed only once.

3. Expected running time

In this section we present a heuristic estimate of the expected running time of the number field sieve. Currently there are various factoring algorithms that have a subexponential expected running time: the continued fraction algorithm, the class group method, the quadratic sieve algorithms, the elliptic curve algorithm, the number field sieve, Dixon's random squares algorithm, Vallee's two-thirds algorithm, and Seysen's class group algorithm (cf. [9]). Only for the last three algorithms a rigorous analysis of the expected running time has been given, for Seysen's algorithm under the assumption of the generalized Riemann hypothesis. These three algorithms tend to be less practical than the other algorithms mentioned above, although for the latter nothing better can be done than a run time analysis that is based on heuristic estimates.

Each of the algorithms mentioned above draws a sequence of integers from a certain distribution. Only those integers that are smooth in a certain sense can be used by the algorithm. Consequently, the expected number of smooth integers in the sequence plays an important role in the running time analysis. A satisfactory estimate of this expected number can be given if each of the integers is uniformly distributed over $[1,B]$, for some upper bound B. However, none of the algorithms mentioned above satisfies this uniformity condition. To obtain a heuristic analysis, one simply *assumes* that the smoothness probabilities are the same as in the uniform case. This can actually be *proved* for the last three algorithms mentioned above, and this leads to a rigorous analysis of their expected running times (modulo the Riemann hypotheses, in one case).

For the other algorithms, including the algorithm described in this paper, nothing better can presently be given than a heuristic analysis, which is better than having nothing at all. Such heuristic analyses add to our understanding of algorithms that are practically useful. In addition, they enable us to compare the algorithms to each other, and to make predictions about their practical performance. If one insists on having fully proved theorems, nothing better can be done than explicitly formulating all heuristic assumptions that enter into the proof. For examples of such theorems we refer to [12]. For the number field sieve we refer to [3].

To analyse the expected running time of the number field sieve we use the following function L from [9, (8.10)]. Let for $\gamma, \nu \in \mathbf{R}$ with $0 \le \nu \le 1$ the function $L_x[\nu;\gamma]$ be any function of x that equals $\exp((\gamma+o(1))(\log x)^{\nu}(\log\log x)^{1-\nu})$, for $x \to \infty$. For fixed $\gamma, \delta, \nu, w \in \mathbf{R}$ with $\gamma, \delta > 0$ and $0 < w < \nu \le 1$, a random positive integer $\le L_x[\nu;\gamma]$ has only prime factors $\le L_x[w;\delta]$ (is $L_x[w;\delta]$-smooth) with probability $L_x[\nu-w; -\gamma(\nu-w)/\delta]$, for $x \to \infty$ (cf. [9, (8.10)]).

Suppose that for a certain $n = r^e - s$ the extension degree d has been chosen close to $(3\log n/(2\log\log n))^{1/3}$. If r and $|s|$ are below a fixed upper bound, it follows that $m = L_n[2/3, (2/3)^{1/3}]$. Furthermore, let $B = L_n[1/3, (2/3)^{2/3}]$, and let a and b both be bounded by $L_n[1/3, (2/3)^{2/3}]$. Notice that $\pi(L_n[1/3, (2/3)^{2/3}]) = L_n[1/3, (2/3)^{2/3}]$. We make the heuristic assumption that $|\mathrm{N}(a+\alpha b)| = |a^d - c(-b)^d|$ and $|a+mb|$, both of which are $\le L_n[2/3, (2/3)^{1/3}]$ if $|c|$ is assumed to be small, behave as random positive integers $\le L_n[2/3, (2/3)^{1/3}]$. This would give each of them a probability $L_n[1/3, -(18)^{-1/3}]$ to be B-smooth. The probability that they are simultaneously B-smooth is therefore assumed to be $L_n[1/3, -(2/3)^{2/3}]$.

It follows that the $L_n[1/3, 2(2/3)^{2/3}]$ pairs a, b can be expected to produce the $L_n[1/3, (2/3)^{2/3}]$ relations of the form (2.4) needed to factor n; this takes expected time $L_n[1/3, 2(2/3)^{2/3}]$. Because the matrix of the exponent vectors is sparse, a dependency can be found in the same amount of time (cf. [16]), so that we expect a running time of $L_n[1/3, 2(2/3)^{2/3}]$ to collect the relations and to derive a factorization from them. The running time is probably only affected by a factor $L_n[1/3, 0]$ if large primes are used as well (see below).

Compared to the collection and elimination steps, the time needed to find U and G is in practice negligible. This would suggest that the total factoring time becomes $L_n[1/3, 2(2/3)^{2/3}] \approx L_n[1/3, 1.526]$. It seems difficult to give a satisfactory asymptotic analysis of the method to find U and G that we described in the previous section. It is not unlikely, however, that alternative methods of finding U and G can be shown to meet the above run time bound.

4. Taking advantage of large primes

A considerable speed-up of the algorithm can be achieved by allowing large primes in (2.1) and (2.2). The use of large primes in factoring algorithms is well known [13]. In the quadratic sieve algorithms relations are collected such that

$$x^2 = q^t \cdot \prod_{p \text{ primes}, p \le B} p^{w_p} \bmod n,$$

for some bound B, integer x, integer $t \in \{0, 1\}$, and prime $q > B$. If more than B of such relations with $t = 0$ (so called *full* relations) are found, the factorization of n can be derived as explained above.

Relations with $t = 1$ (*partial* relations) are, however, much easier to find than full relations. Furthermore, two partial relations with the same q can be combined (i.e., multiplied) into one relation that is equally useful (but denser, see below) for the factorization process as a full relation. Because the partials come in much faster than the fulls, one expects to find quite a few double q's (cf. birthday paradox), and consequently quite a few additional useful relations. In practice this leads to a speed-up by more than a factor of two: for a certain n for which we used $B = 50{,}000$, only 20,000 of the first 320,000 relations were full. But the remaining 300,000 partials sufficed to generate the other 30,000 relations needed for the factorization (cf. [10]).

In (2.1) and (2.2) the situation is similar, but allows for more variations. In the first place we notice that there is no problem at all in allowing a large prime at the right hand side in (2.2), but no large prime in (2.1). It leads to a relation as in (2.4) with a large prime at the right hand side, a so-called *fp*, for *full-partial*-relation. Two *fp*'s with the same large prime can be combined into a relation that is as useful as (2.4).

A large prime at the right hand side of (2.1), but no large prime in (2.2), a *pf*, leads to a slightly more complicated situation. To be able to write down (2.4) we would need a generator of the prime ideal corresponding to the large prime (i.e., of the same norm, and having the same root modulo the large prime as the corresponding $a+\alpha b$). Although such a generator would only be needed for large prime ideals that will be matched, this still does not look very appealing, at least not given our way of finding generators. Fortunately, there is an easy way out of this problem: combine relations with the same large prime ideal by dividing them, instead of multiplying them. In that way generators corresponding to the large primes are not needed, but two *pf*'s with the same large prime ideal can be combined as before into a useful relation. The only difference is that we now have to allow for negative exponents v_g, $\bar{v}_g$, w_p, and $\bar{w}_p$ in (2.4) and (2.7).

The *fp*'s and the *pf*'s already allow us to take advantage of partial relations in a way that is similar to the quadratic sieve algorithms. But we can do even more by taking the *pp*'s, the relations having a large prime both in (2.1) and in (2.2), into account as well. For example, the product of an *fp* with large prime q_1 and a *pf* with large prime q_2 divided by a *pp* with large primes q_2 (in (2.1)) and q_1 (in (2.2)) is a useful relation (if the prime ideals of norm q_2 are the same). Notice that combinations do not necessarily have to begin and end with an *fp* or *pf*; cycles among the *pp*'s also lead to useful relations.

If the *fp*'s and *pf*'s are easy to find compared to the *ff*'s (the full-full's, as in (2.4)), then the *pp*'s should be even easier to find (see Section 6 for an example of actual fractions between numbers of *ff*'s, *fp*'s, *pf*'s, and *pp*'s). Intuitively it should be the case that, if the *fp*'s have already a reasonable probability to match among themselves, then they will have a much higher probability to match with the *pp*'s; the *fp,pp* pairs thus found should then still have a reasonable probability to match with the *pf*'s, and a somewhat higher probability to match with the remaining *pp*'s. That this indeed works in practice can be seen in Section 6.

While generating the combinations, care should be taken that they remain independent, the extreme example being that an *fp* can but should not be combined with itself. In the full version of this paper we will describe our algorithm to find a maximal independent set of combined relations, including cycles among the *pp*'s. Our algorithm cannot be guaranteed to find the shortest combinations, but the combinations will never be more than twice too long.

For relations that use *fp*'s, *pf*'s and *pp*'s the unit contribution can easily be found by adding and/or subtracting the relevant low dimensional vectors that were introduced above. As noted above the combined relations give rise to much denser rows of the matrix than the *ff*'s. This makes the matrix elimination step much slower.

5. Additional remarks

There is yet another way to get some extra relations, the *free* relations. Suppose that for some prime $p \leq B$ the polynomial $f \bmod p$ factors completely into linear factors over $\mathbf{Z}/p\mathbf{Z}$. Then the ideal generated by p is equal to the corresponding product of the prime ideals of norm p. Hence p equals a unit times a product of the generators of those ideals. This unit can be determined as above. The density of the primes that split completely in this way is the inverse of the degree of the normal closure of K, which divides $d \cdot \phi(d)$ and is a multiple of $\mathrm{lcm}(d, \phi(d))$ (with ϕ the Euler ϕ-function). So, for $d = 5$, we get about $\#G/20$ relations for free, which amounts to about 1/40 of the relations needed.

It is of course not at all necessary to have two factor bases of approximately the same size, although it is asymptotically optimal. For any particular number n and choice of d it might be advantageous to select a bigger factor base on either side. Sizes that work satisfactorily can usually quite easily be found experimentally.

Another idea that might be of practical value is to use more than one number field. If we use smoothness bound B_i in (2.1) for the ith number field K_i, and for all i's the same smoothness bound B in (2.2), then we need approximately $\pi(B)+\sum_i(\#U_i+\#G_i)$ relations (2.4) to factor n, where U_i and G_i generate the units and the prime ideals of norms $\leq B_i$ in K_i. So, each K_i should contribute at least $\#U_i+\#G_i$ relations to make using the ith field worthwhile. The extension degrees of the number fields

do not have to be distinct, as the following example shows. Let ζ be a 16th root of unity, then the number fields $\mathbf{Q}(\zeta^2)$, $\mathbf{Q}(\zeta-\zeta^{-1})$, and $\mathbf{Q}(\zeta+\zeta^{-1})$, all of extension degree 4, could all be used for the factorization of $2^{512}+1$.

We do not have any practical experience yet with this multi-field approach. We expect that it is another way to lower the total factor base size: for each new field we can start afresh with small b values, which have a higher probability of success.

6. Results

The first factorization obtained by means of the number field sieve (nfs) was the (already known, cf. [11]) factorization of the 39 digit number $F_7 = 2^{2^7}+1$. This factorization was carried out by the fourth author in twenty hours on a Philips P2012, an 8-bit computer with 64K of memory and two 640K disc drives. With $f(X) = X^3+2$, one unit and 497 prime ideals for the factorizations in (2.3), and 500 primes for (2.1), it took 2000 b's (and $a \in [-4800,4800]$) to find 538 *ff*'s and 1133 *fp*'s with large prime < 10,000 (no *pf*'s or *pp*'s were used). This led to 399 combinations, which combined with the 81 free relations sufficed to factor F_7:

$$2^{128}+1 = 59649589127497217 * 5704689200685129054721.$$

Several steps of this first nfs factorization were not carried out as described in the previous sections. For instance, only the $a+mb$'s were being sieved, and for the reports both $a+mb$ and $N(a+\alpha b)$ were tested for smoothness. The unit contribution was found by means of a table containing u_1^i for $i = -8, ..., 8$. 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 Section 2. 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):

$$2^{144}-3 = 492729991333 * 4525956526047789916201098027276 1,$$

$$2^{153}+3 = 5 * 11 * 600696432006490087537 * 3455982977796034189382757.$$

For numbers in the 100+ range the nfs can be expected to run faster than the multiple polynomial quadratic sieve algorithm (mpqs), at least when applied to numbers of the right form. But one still needs quite impressive computational resources to factor numbers that large.

In our implementation at Digital Equipment Corporation's Systems Research Center (SRC) we followed the same approach as in our SRC-implementations of the elliptic curve method (ecm) and mpqs as described in [10]. In short, this means that one central processor distributes tasks among several hundred CVAX processors, the clients, and collects their results.

For nfs tasks consist of short, non-overlapping intervals of consecutive b values. When given an interval $[b_1, b_1+1, ..., b_2)$, a client starts sieving all pairs a, b for $|a|$ less than some predetermined bound, and for $b = b_1, b_1+1, ..., b_2-1$ in succession. After each b, a client reports the good pairs a, b to the central processor, and it reports that it just processed that particular b. The central processor keeps track of the good pairs it received, and of the b's that have been processed. It also notices if a client dies or becomes unavailable (for instance because the owner claims his machine), so that it can redistribute the b's that are left unfinished by that client. In this way, all positive b's will be processed, without gaps, until sufficiently many good pairs have been collected.

This is slightly different from our ecm and mpqs implementations where we do not worry at all about inputs that have been distributed but that have never been processed completely. For ecm and mpqs we could easily afford that. For nfs it might be possible as well, but because smaller b's are better than larger b's we decided to be careful and not to waste any inputs.

In Table 1 we list some of the new results obtained so far. For the factorization of the first two entries we did not make use of the *pp*'s yet. Clearly, both these numbers could have been factored with much smaller factor bases had we used the *pp*'s as well. The first entry was the first number we collected relations for. Still being quite unexperienced with the method, we chose the factor bases for that number much too big, even without using the *pp*'s. In the 'run-time' columns it should be kept in mind that the relations were collected on a network of several hundred CVAX-es, but the elimination was done in parallel on one single machine containing six CVAX-es. For all factorizations in Table 1 we sieved for each b over the a in $[-5\cdot10^6, 5\cdot10^6)$, split up in intervals of length 500,000. The limit for the large primes was 10^8. For the last three cases we found that $\mathbf{Z}[\alpha]$ is not a unique factorization domain, a problem that was easily overcome because in all cases the full ring of integers of K does have unique factorization.

For the first two entries about 2/5 of the relations needed were *ff*'s and the remaining 3/5 were split evenly among the *fp*, *fp* pairs and *pf*, *pf* pairs. For the third entry we had 10,688 *ff*'s, 103,692 *fp*'s, 116,410 *pf*'s, and 1,138,617 *pp*'s after 1,136,000 b's. This gave 5,058 *fp*, *fp* pairs, and 5,341 *pf*, *pf* pairs. Furthermore we had 1,222 free relations of the type $\phi(u\cdot\prod g) = p$ for each prime $p \le B$ for which $f \bmod p$ has d roots, where the product ranges over d generators g of norm p and where u is a unit. About 28,000 additional relations were required, and these were obtained from more complicated combinations containing *pp*'s. Because the yield was already quite low for b around 1,100,000, we would never have been able to factor that number with those factor base sizes had we not used the *pp*'s as well.

For the last entry we had gathered 17,625 *ff*'s, and a

total of 1,741,365 *fp*'s, *pf*'s, and *pp*'s, which gave 62,842 combinations. Furthermore we had 2,003 free relations.

For all numbers in Table 1 the set U contained only -1 and two units which were easy to find. Finding G never took more than about fifteen minutes on a CVAX processor.

Notice that the relation collection stage for $2^{457}+1$ took 7/2 times as much time as for $7^{149}+1$. This is approximately the same as (80,000/50,000)*(2,650,000/1,136,000) and also approximately the same as we expect from the run time analysis.

7. Generalization

To generalize the number field sieve to general integers, i.e., integers which are not of the form r^e-s for small r and s (up to a small factor), it suffices to select some integer d, an integer m close to and at most $n^{1/d}$, and to put $f(X)=\sum_{i=0}^{d} f_i X^i$, where $n=\sum_{i=0}^{d} f_i m^i$ with $0 \le f_i < m$. Now use $K=\mathbf{Q}(\alpha)$ with $f(\alpha)=0$. This was suggested by Joe Buhler and Carl Pomerance.

In principle the algorithm could proceed as described above. There are some serious problems, however, that make part of that approach unfeasible, and for which a satisfactory solution is still unknown. For number fields with a relatively small discriminant as in Section 2, the sets U and G are not at all difficult to construct. If the discriminant gets bigger, as will in general be the case for the polynomial f as constructed here, our method will not work: the values for a_B and C_B 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.

It follows that actual computation of U and G should be avoided. This can be achieved as follows. Suppose that sufficiently many good pairs a, b have been found. So, for each good pair a, b, we can write the ideal $(a+\alpha b)$ as the product of prime *ideals* g of norms $\le B$,

$$(a+\alpha b)=\prod_g g^{v_g},$$

and the integer $a+mb$ is B-smooth,

$$a+mb=\prod_{p \text{ prime}, p \le B} p^{w_p}$$

(where -1 is among the p's). For simplicity, we assume that K has an embedding into $\mathbf{R}$ for which all $a+\alpha b$ are positive.

Let M be a matrix that contains, for each good pair a, b, a row consisting of the concatenation of the vectors (v_g) and $(w_p \bmod 2)$, and let h be an integer slightly bigger than l, where l is as in Section 2. One now finds h independent relations, with integer coefficients, between the rows of M; in the left half the relations should be valid in $\mathbf{Z}$, in the right half they need only be valid in $\mathbf{Z}/2\mathbf{Z}$. This yields, for each i with $1 \le i \le h$, integers $x_i(a, b)$ such that at the same time

$$\prod_{a,b}(a+\alpha b)^{x_i(a,b)}=(1)$$

as ideals, and

$$\prod_{a,b}(a+mb)^{x_i(a,b)}=\Big(\prod_{p \text{ prime}, p \le B} p^{\bar{w}_p}\Big)^2,$$

Table 1
New factorizations obtained with the number field sieve

n is factor of	# digits of n	$f(X)$	sizes factor bases	used b up to	# digits of factors	run-time relations	run-time elimination
$3^{239}-1$	107	X^5-3	2*40,000	120,000	41 & 67	two days	two weeks
$2^{373}+1$	108	X^5+4	2*25,000	200,000	48 & 60	three days	four days
$7^{149}+1$	122	X^5+7	2*25,000	1,136,000	56 & 66	two weeks	five days
$2^{457}+1$	138	X^5+8	2*40,000	2,650,000	49 & 89	seven weeks	two weeks

$3^{239}-1 = 2 * 479 * 17209 * 4330196405563553333394574553310612804421 3 *$
$150782569243713367195998138893556498577755606347207456007344127343 1;$

$2^{373}+1 = 3 * 60427 * 69457949731689426442566124365980637197218831885 7 *$
$152796756325290043462779779478758328705905947521327614399129;$

$7^{149}+1 = 8 * 10133 * 473384333551899292791106509318378061198290085739285016 23 *$
$216553920907130765514882971236304435176465908770055947615503538839;$

$2^{457}+1 = 3 * 68853575602053195730606338968009184482549047291 93 *$
$180160784963046309016951710213374310269101720810721770473272722133609167604597928093055 87.$

where $\bar{w}_{ip} \in \mathbf{Z}$. This leads to a relation of the form

$$\phi(u_i) \equiv \Big(\prod_{p \text{ prime}, p \le B} p^{\bar{w}_{ip}}\Big)^2 \bmod n,$$

where u_i is a unit that can be written as $\prod_{a,b}(a+\alpha b)^{x_i(a,b)}$. Use this expression to compute the vectors $v(u_i) \in \mathbf{R}^l$, where the mapping v is as in Section 2.

Each of the the $h > l$ vectors $v(u_i)$ is contained in the same l-dimensional lattice, so that a $\mathbf{Z}$-relation between them can be found by means of basis reduction. The corresponding product of the u_i's then leads to a solution of $1 \equiv z^2 \bmod n$.

The most recent heuristic estimate of the expected running time of the relation collection and matrix elimination stages of the generalized algorithm is $L_n[1/3, 3^{2/3}]$. It is expected that the most serious practical problems with the generalized algorithm will be caused by the elimination over $\mathbf{Z}$ and the size of the integers involved.

References

1 Red Alford, C. Pomerance, personal communication.

2 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*, Contemporary Mathematics, vol. 22, Providence: A.M.S., 1988.

3 J. Buhler, H.W. Lenstra, Jr., C. Pomerance, in preparation.

4 T.R. Caron, R.D. Silverman, "Parallel implementation of the quadratic sieve," J. Supercomputing, v. 1, 1988, pp 273-290.

5 D. Coppersmith, A.M. Odlyzko, R. Schroeppel, "Discrete Logarithms in $GF(p)$," Algorithmica, v. 1, 1986, pp 1-15.

6 D.E. Knuth, "Computer Science and its relation to mathematics," Amer. Math. Monthly, v. 81, 1974, pp 323-342.

7 D.E. Knuth, *The art of computer programming*, vol. 2, *Seminumerical algorithms*, second edition, Addison-Wesley, Reading 1981.

8 S. Lang, *Algebra*, second edition, Addison-Wesley, Reading, 1984.

9 A.K. Lenstra, H.W. Lenstra, Jr., "Algorithms in number theory," to appear in: J. van Leeuwen, A. Meyer, M. Nivat, M. Paterson, D. Perrin (eds), *Handbook of theoretical computer science*, North-Holland, Amsterdam.

10 A.K. Lenstra, M.S. Manasse, "Factoring by electronic mail," Proceedings Eurocrypt '89, to appear.

11 M.A. Morrison, J. Brillhart, "A method of factoring and the factorization of F_7," Math. Comp., v. 29, 1975, pp 183-205.

12 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 1982.

13 C. Pomerance, S.S. Wagstaff, Jr., "Implementation of the continued fraction integer factoring algorithm," Congress. Numer., v. 37, 1983, pp 99-118.

14 H.J.J. te Riele, W.M. Lioen, D.T. Winter, "Factoring with the quadratic sieve on large vector computers," report NM-R8805, 1988, Centrum voor Wiskunde en Informatica, Amsterdam.

15 I.N. Stewart, D.O. Tall, *Algebraic number theory*, second edition, Chapman and Hall, 1987.

16 D.H. Wiedemann, "Solving sparse linear equations over finite fields," IEEE Trans. Inform. Theory, IT-32, 1986, pp 54-62.