Factoring Large Integers with the Quadratic Sieve

Factoring Large Integers with the Quadratic Sieve

PROEFSCHRIFT

TER VERKRIJGING VAN DE GRAAD VAN DOCTOR
AAN DE RIJKSUNIVERSITEIT TE LEIDEN,
OP GEZAG VAN DE RECTOR MAGNIFICUS
DR. W. A. WAGENAAR,
HOOGLERAAR IN DE FACULTEIT DER SOCIALE
WETENSCHAPPEN,
VOLGENS BESLUIT VAN HET COLLEGE VAN
DEKANEN
TE VERDEDIGEN OP DINSDAG 10 JUNI 1997
TE KLOKKE 14:15

DOOR

HENDRIK BOENDER
GEBOREN TE ROTTERDAM IN 1965

Samenstelling der promotiecommissie:

promotor:	prof. dr. R. Tijdeman
copromotor:	dr. ir. H. J. J. te Riele (CWI)
referent:	dr. B. M. M. de Weger (EUR en SWON/NWO)
overige leden:	prof. dr. ir. L. A. Peletier
	prof. dr. M. N. Spijker
	prof. dr. L. C. M. Kallenberg (RUG)

Contents

List of symbols

B ..... upper bound for the elements in the factor base
Cx ..... a composite number with x decimal digits
$\mathcal{F}$ ..... factor base
k ..... number of different g–primes in the leading coefficient of $W(x)$
L ..... upper bound for the large primes in MPQS1
L^2 ..... upper bound for the product of the large primes in MPQS2
m ..... multiplier
M ..... radius of the sieve interval $[-M, M]$
n ..... number to be factored (contains multiplier)
n_c ..... number of complete relations found immediately
$n_{c,1}$ ..... number of complete relations from the partial relations
$n_{c,2}$ ..... number of complete relations generated by combining the partial relations with different large primes and the partial–partial relations
n_f ..... number of elements in the factor base
n_1 ..... number of partial relations found
n_2 ..... number of partial–partial relations found
$\left(\frac{n}{p}\right)$ ..... Legendre symbol
$\mathbf{N}$ ..... $\{1, 2, 3, \ldots\}$
$\pi(y)$ ..... the number of primes $\leq y$
$\psi(x, y)$ ..... the number of positive y–smooth integers $\leq x$
QT ..... upper bound for primes we ignore in the sieve part
r ..... number of generated g–primes in Chapter 1, 2, and 3
r ..... number of sieved polynomials in Chapter 4
R ..... factor of $W(x)$ that does not contain primes of the factor base in Chapter 1, 2, and 3
R ..... $-R$ is the minimum of $W(x)$ on the sieve interval in Chapter 4
RT ..... report threshold
S ..... sieve array
S ..... maximum of $W(x)$ on the sieve interval
T_s ..... sieve time
$W(x)$ ..... $W(x) = a^2x^2 + 2bx + c$ in Chapter 1, 2, and 3
$W(x)$ ..... $W(x) = ax^2 - 2bx - c$ in Chapter 4
$\mathbf{Z}$ ..... $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$

Chapter 1

Introduction

1.1 Factoring integers

Books VII, VIII and IX of the famous *Elements* by Euclid (300 B.C.) are devoted to the theory of numbers. Euclid recorded the Fundamental Theorem of Arithmetic in Proposition IX, 14, the modern statement of which is:

Every integer greater than 1 is a prime or can be written as a product of primes in exactly one way, except for the ordering of the factors.

Euclid proved the theorem only for the class of numbers that consist of a product of *different* primes [9, p. 191]. Decomposing an integer n into its prime factors is an iterative process. If one has found a non–trivial factor m of n, one proceeds with m and the cofactor n/m. Therefore we say that we have *factored* n if we have found a non–trivial divisor of n. It does not matter *how* the computations are carried out since the result is easy to check: just divide n by a putative factor and see whether the result is an integer. It is known for several thousands of years that decomposing an integer into factors is doable in principle. Actually factoring large sample numbers is another story.

1.1.1 Suggestions for reading on factoring

Williams and Shallit [47] have given a history of factoring and primality testing from 1750 to 1950, i.e., before the era of electronic computers. Richard Guy [17] has written a good survey of factorization methods known in 1975. Brillhart et al. [8] have presented a chronology of developments in factorization, both hardware and software, esp. the methods used by the Cunningham project (see Appendix). A more recent exposition is given by Robert Silverman [45]. Several textbooks [7, 11, 20, 42] cover factorization.

Peter L. Montgomery has presented a survey of modern integer factorization algorithms in [28]. Some of this material is used in this chapter. Andrew Odlyzko [33] has taken a cautious glimpse in the crystal ball.

1.1.2 The limit in the course of years

In 1970 it was barely possible to factor general form 20–digit numbers (here a d–digit number is a number with d *decimal* digits, unless specified otherwise). Factoring 50–digit numbers became commonplace in 1980 and in 1990 the record was 116 digits [39]. In Martin Gardner's 1976 Scientific American column the 129–digit RSA challenge number (see Section 1.2.2) had been estimated to be safe for 40 quadrillion years. Whether it were American or British quadrillion years, it does not matter any more: eighteen years later the number had been factored with the help of hundreds of persons and their computers all over the world [2]. The current record (at 1st May 1997) is a 130–digit RSA challenge number, split in the spring of 1996 [12]. The progress in factoring integers is due to better algorithms, the availability of more and faster computers, the invention of parallel computers, and the use of network computers (factoring via Internet).

1.2 Motivation

Before 1978 factoring integers was mainly of academic and recreational interest. In 1978 however factoring was put into the limelight by three men, R. L. Rivest, A. Shamir, and L. Adleman, who invented a public–key cryptosystem whose safety is based upon the practical difficulty of decomposing large integers into prime factors.

1.2.1 Cryptosystems

A cryptosystem is a protocol to transform a clear text message into a text that is unreadable for an eavesdropper when it is sent to a receiver. Naturally, the protocol supplies the receiver with a way to decipher the message into its original form. Often the sender and receiver agree upon a secret key before the transmission takes place. With such a key the receiver can trace back the original text. The key has to be transported via a safe channel. The Dutch Ministry of Foreign Affairs, e.g., sometimes uses a courier service for this. Every now and then the key has to be "refreshed" to guarantee safety for a certain period of time. It would be handy to agree upon a key that may be transported via a safe channel. This is when a public–key cryptosystem comes in.

1.2.2 The RSA cryptosystem

A public–key cryptosystem provides every user with two personal keys: one publicly known key and one key that is only known to the user it belongs to. Suppose that Alice wants to send an encrypted message to Bob. She looks up Bob's key in, say, a kind of telephone directory and encrypts the message with his key. Bob then decrypts the message he receives with his secret key. A public–key cryptosystem must fulfil the following requirements:

- Each user has an own publicly known encryption procedure E and a private decryption procedure D.
- For all messages M we have $D(E(M)) = E(D(M)) = M$.
- D and E are cheap to execute.
- It is infeasible to find D given only E.

The RSA public–key cryptosystem enables this. Each user gets two primes $p \neq q$ (no user gets a prime that another user already has) and a so–called exponent e that has no factors in common with $(p-1)(q-1)$. Let $n = p \cdot q$. The user computes d such that $de \equiv 1 \pmod{(p-1)(q-1)}$. The public key is the pair (e, n) and the secret key is the triple (d, p, q). Let the message space be $[0, n-1]$ (split a clear text into chunks so that it is possible to decode each chunk into a number in the message space). The encryption $E(M)$ of a message $M \in [0, n-1]$ is defined by $E(M) \equiv M^e \pmod{n}$ and to decrypt a message M one has to compute $D(M) \equiv M^d \pmod{n}$ (both $E(M)$ and $D(M)$ lie in the message space).

If one can factor n, then one has the prime factors p and q. Since e is public, it is then easy to compute d and thus the secret key. Hence, if one has an algorithm that quickly decomposes a general form number into its prime factors, then one can break RSA in a reasonable amount of time. So factoring is sufficient. Faster ways to break the code are unknown. Therefore factoring is important. The reverse is unclear: it is not known whether one can factor easily if one can break RSA easily.

1.3 Factoring with differences of squares

Let n be an odd positive integer to be factored and suppose that n is not a prime power. See algorithm 1.7.4 in [11] for prime power detection. (But do not forget the errata et addenda.) Fermat noticed that if we have two integers X and Y such that $X^2 - Y^2 = n$, then n is equal to $(X-Y)(X+Y)$

so that n is factored if $X - Y$ is a non–trivial divisor of n. If n has two factors that lie close to the square root of n, then it is easy to find the two squares. Most large numbers however have factors that are not near to the square root. Hence the difference–of–squares method is not practical for large numbers. In the 1920s Maurice Kraitchik argued that it might suffice to find X and Y such that the difference of their squares is a multiple of n:

$$X^2 \equiv Y^2 \pmod{n}. \tag{1.1}$$

Then the greatest common divisor of $X - Y$ and n is a non–trivial factor of n if $X \not\equiv \pm Y \pmod{n}$. If X and Y are randomly chosen subject to (1.1), then this yields a proper factor of n in at least 50 % of the tries. This principle is the basis for the best known general factorization methods, namely, the multi–polynomial quadratic sieve (MPQS [1, 2, 5, 14, 15, 16, 25, 36, 37, 38, 40, 41, 44]) and the number field sieve (NFS [24]). The continued fraction method CFRAC of Morrison and Brillhart [31] is also based on this principle. It is the precursor of the quadratic sieve (QS). In the next section we give a method to find X and Y.

1.3.1 Constructing a square with smooth numbers

Let m be an integer. We write $P(m)$ for the largest positive prime factor of m. If $P(m) \leq y$ then m is called *y–smooth.*

To find X and Y we proceed as follows. Choose a smoothness bound B beforehand. Look for squares X_i^2 such that the remainder $r_i = X_i^2 \pmod{n}$ is $\mathcal{O}(\sqrt{n})$ $(i = 1, 2, \ldots)$. If r_i is B–smooth, then save the ith congruence, otherwise reject it.

In Sections 1.3.4 and 1.4 we give methods to construct the numbers X_i and r_i. Here we show how to construct X and Y from the saved congruences. Let $\{p_1, p_2, \ldots, p_\ell\}$ be the set of primes $\leq B$ and let $p_0 = -1$ for convenience. Suppose we saved $t > \ell + 1$ congruences

$$X_i^2 \equiv r_i \pmod{n} \quad (i = 1, 2, \ldots, t).$$

Since r_i is B–smooth, it splits completely over the set of primes $\leq B$:

$$r_i = \prod_{j=0}^{\ell} p_j^{e_{i,j}},$$

where $e_{i,j} \geq 0$ is the exponent of prime p_j in r_i if $j > 0$. We define $e_{i,0} = 0$ if $r_i \geq 0$ and $e_{i,0} = 1$ if $r_i < 0$. Determine a subset $\mathcal{S} \subset \{1, 2, \ldots, t\}$ such

that the product of the r_i's, where i runs through $\mathcal{S}$, is a square. We then have

$$\prod_{i\in\mathcal{S}} X_i^2 \equiv \prod_{i\in\mathcal{S}} r_i \pmod{n}.$$

Now we may take

$$X = \prod_{i\in\mathcal{S}} X_i \quad \text{and} \quad Y = \sqrt{\prod_{i\in\mathcal{S}} r_i}.$$

How do we determine a subset $\mathcal{S}$ with the desired property? Of course we are done if one of the r_i's is a square, but this is a rare occasion. If we multiply r_i's, the exponents of the same factors of the r_i's add. The product is a square precisely when all the exponents sum to an even number. Thus it makes sense to consider the set of *exponent vectors* mod 2:

$$(e_{i,j} \pmod 2) \quad (i = 1, 2, \ldots t, \quad j = 0, 1, \ldots \ell).$$

The set forms a matrix in which row i contains the exponents mod 2 of the p_j's in r_i. Seeking a suitable subset $\mathcal{S}$ corresponds to the search for a set of rows that sum to zero mod 2. Since there are more rows (t) than columns ($\ell + 1$), we are sure that such a set of rows exists. We may find it, e.g., by using Gaussian elimination mod 2.

1.3.2 Finding dependencies modulo 2

Factoring a number of more than 100 digits, say, requires a large smoothness bound to find sufficiently many smooth numbers in a reasonable amount of time. Thus the resulting matrix is large (about $10^5 \times 10^5$ typically), but sparse (about 20 non–zero entries per row typically). When we use Gaussian elimination, the matrix transforms during the process and many entries that were zero at the beginning change to one. This is called *fill–in*. Consequently, at the end of the Gaussian elimination phase almost every entry of the matrix must be stored, which requires a huge amount of memory. (If there are only few ones, that is if the matrix is sparse, then there are more efficient ways to store the matrix.)

To avoid fill–in we may use the Lanczos method [22] or rather the block Lanczos method [30]. The latter one is designed especially for factorization purposes by Peter L. Montgomery at Centrum voor Wiskunde en Informatica (CWI) in Amsterdam. The Lanczos methods are so–called iterative methods that repeatedly require matrix–vector and vector–vector (dot product) operations, but leave the matrix unimpaired.

1.3.3 Intermezzo: continued fractions

We treat continued fractions in a nutshell. For further reading and proofs we refer to Chapter 10 of [19].

For a real number α we define

$$\begin{aligned}
\alpha &= a_0 + \alpha_1 \quad (a_0 \in \mathbf{Z},\ 0 \leq \alpha_1 < 1),\\
\tfrac{1}{\alpha_1} &= a_1 + \alpha_2 \quad (a_1 \in \mathbf{Z},\ 0 \leq \alpha_2 < 1),\\
&\vdots\\
\tfrac{1}{\alpha_{k-1}} &= a_{k-1} + \alpha_k \quad (a_{k-1} \in \mathbf{Z},\ 0 \leq \alpha_k < 1),
\end{aligned}$$

and we stop if the remainder α_j is zero. We then have the *continued fraction* of α:

$$\alpha = a_0 + \cfrac{1}{a_1 + \cfrac{1}{a_2 + \cfrac{1}{\ddots\, a_{k-2} + \cfrac{1}{a_{k-1} + \alpha_k}}}}.$$

We write $\alpha = [a_0, a_1, a_2, \ldots, a_{k-2}, a_{k-1} + \alpha_k]$. It is clear that $a_j \in \mathbf{N}$ for $j \geq 1$. The continued fraction of α is finite, i.e., there is a remainder α_j that is zero, if and only if α is rational. Here we assume that α is irrational.

We define two sequences $\{p_k\}_{k=-1}^{\infty}$ and $\{q_k\}_{k=-1}^{\infty}$ as follows:

$$\begin{aligned}
p_{-1} &= 0, \quad p_0 = 1, \quad p_k = a_{k-1}p_{k-1} + p_{k-2},\\
q_{-1} &= 1, \quad q_0 = 0, \quad q_k = a_{k-1}q_{k-1} + q_{k-2}
\end{aligned}$$

$(k = 1, 2, \ldots)$.

With an induction argument it is easy to prove that we have

$$[a_0, a_1, a_2, \ldots, a_{k-2}, a_{k-1}] = \frac{p_k}{q_k} \quad (k = 1, 2, \ldots).$$

Another identity is

$$p_k q_{k-1} - p_{k-1} q_k = (-1)^k \quad (k = 0, 1, \ldots)$$

so that the fractions $\frac{p_k}{q_k}$, called the *convergents* of α, are reduced.
The convergents are very good rational approximations to the irrational number α:

$$\left| \frac{p_k}{q_k} - \alpha \right| < \frac{1}{q_k^2} \quad (k = 1, 2, \ldots). \tag{1.2}$$

(Note that the sequence $\{q_k\}_{k=2}^{\infty}$ is strictly increasing.) From this it follows that the sequence $\left\{\frac{p_k}{q_k}\right\}$ is convergent and tends to α if $k \to \infty$.

1.3.4 Description of CFRAC

We now explain how to generate the numbers X_i and r_i with CFRAC. Of course we may assume that n is not a perfect square so that $\sqrt{n}$ is irrational. Hence there exist infinitely many rational approximations p/q of $\sqrt{n}$ with $\gcd(p, q) = 1$ and such that

$$\left| \frac{p}{q} - \sqrt{n} \right| < \frac{1}{q^2}.$$

This follows from (1.2). If p/q is any such approximation, write $p/q = \sqrt{n} + \epsilon/q^2$, where $|\epsilon| < 1$. We have

$$p^2 - nq^2 = \left(q\sqrt{n} + \frac{\epsilon}{q} \right)^2 - nq^2 = 2\epsilon\sqrt{n} + \frac{\epsilon^2}{q^2}$$

and thus

$$\left| p^2 - nq^2 \right| < 2\sqrt{n} + \frac{1}{q^2} = \mathcal{O}\left(\sqrt{n}\right).$$

For the fractions p/q we may take the convergents in the continued fraction expansion of $\sqrt{n}$. (The convergents of $\sqrt{n}$ may be computed using integer arithmetic [42, pp. 315–317].) Now, if p/q is the ith convergent generated, we define $X_i = p$ and $r_i = p^2 - nq^2$ so that indeed $X_i^2 \equiv r_i \pmod{n}$ and $r_i = \mathcal{O}(\sqrt{n})$.

The latter property just tells us that $|r_i|$ is small compared with n and hence that the probability that r_i is B–smooth is much larger than the smoothness probability of numbers of size n. If n becomes large, however, then $|r_i|$ becomes large too so that the smoothness probability of r_i drops. Therefore the algorithm is only applicable for numbers n that are not too large. To be specific, in 1970 the seventh Fermat number $2^{2^7} + 1$ (39 digits) was split with CFRAC [31, p. 184] and this factorization settled a record. At present CFRAC is not competitive with modern factoring methods.

1.3.5 An abortive attempt

We elucidate the continued fraction method with an example that can also be found in [28]. The sample number is $n = 1098413$; it is the concatenation of the zip code and street address of CWI. We choose $B = 19$. This choice is arbitrary. In Table 1.1 we list convergents p/q of $\sqrt{n}$ and the factorizations of $p^2 - nq^2$.

convergent p/q	$p^2 - n \cdot q^2$
1048/1	$-109 = -109$
19913/19	$476 = 2^2 \cdot 7 \cdot 17$
80700/77	$-677 = -677$
181313/173	$1292 = 2^2 \cdot 17 \cdot 19$
262013/250	$-331 = -331$
1491378/1423	$1207 = 17 \cdot 71$
1753391/1673	$-796 = -2^2 \cdot 199$
3244769/3096	$1153 = 1153$
4998160/4769	$-493 = -17 \cdot 29$
18239249/17403	$1084 = 2^2 \cdot 271$
23237409/22172	$-911 = -911$
41476658/39575	$839 = 839$
64714067/61747	$-1228 = -2^2 \cdot 307$
106190725/101322	$133 = 7 \cdot 19$

Table 1.1: Approximations to $\sqrt{1098413}$.

Three B–smooth values of $p^2 - nq^2$ are

$$\begin{aligned} 19913^2 - n \cdot 19^2 &= 476 = 2^2 \cdot 7 \cdot 17, \\ 181313^2 - n \cdot 173^2 &= 1292 = 2^2 \cdot 17 \cdot 19, \\ 106190725^2 - n \cdot 101322^2 &= 133 = 7 \cdot 19. \end{aligned}$$

The product of the three right sides is a square, namely $2^4 \cdot 7^2 \cdot 17^2 \cdot 19^2$. Multiply the three left sides and suppress multiples of n to reveal

$$(19913 \cdot 181313 \cdot 106190725)^2 \equiv \left(2^2 \cdot 7 \cdot 17 \cdot 19\right)^2 \pmod{n}.$$

Reducing modulo n yields $9044^2 \equiv 9044^2 \pmod{n}$. Unfortunately, this trivial congruence does not yield a proper factorization. We could try more, but leave the reader waiting in tension until Section 1.4.2.

The example illustrates that the continued fraction method, as are the quadratic– and number field sieve, is a so–called probabilistic algorithm: it might fail. A deterministic algorithm on the other hand always yields a factorization in the long run, but in practice such a method is more expensive than a probabilistic one. In practice the continued fraction method and the other methods mentioned, yield a factorization after a few attempts, provided that enough smooth numbers have been found.

1.4 The quadratic sieve and variants

The father of the quadratic sieve (QS) is Carl Pomerance [37] who was inspired by the work of Kraitchik [21] who, on his turn, used ideas of Seelhoff [43]. Based upon ideas of Richard Schroeppel, Pomerance introduced the concept of sieving. Jim Davis and Diane Holdridge [14] were the first to extend QS to a multi–polynomial variant (MPQS) which was an important enhancement. The large prime variants have their roots in CFRAC. Alford and Pomerance [1] came up with the self–initializing variant.

1.4.1 One polynomial

Just as the continued fraction method, the quadratic sieve looks for congruences $X_i^2 \equiv r_i \pmod{n}$ where r_i is smooth. A simple way to generate congruences with a square on one side already is to evaluate a quadratic polynomial in successive points. An example illustrates the procedure.

Again, let $n = 1098413$ and define $f(x) = x^2 - n$. Small values of f lie near the zero $\sqrt{n} = 1048.052\ldots$ Figures 1.1 and 1.2 show 29–smooth polynomial values and the associated binary matrix (here the exponent vectors form the columns instead of the rows). Since -1 plays the same role as prime factors, we also use the letter p to denote it.

$$\begin{array}{|l|}\hline f(\ 925) = -2^2 \cdot 7 \cdot 13 \cdot 23 \cdot 29 \\ f(1047) = -2^2 \cdot 19 \cdot 29 \\ f(1051) = 2^2 \cdot 7 \cdot 13 \cdot 17 \\ f(1063) = 2^2 \cdot 7^3 \cdot 23 \\ f(1077) = 2^2 \cdot 7 \cdot 13^3 \\ f(1119) = 2^2 \cdot 7 \cdot 17^2 \cdot 19 \\ f(1142) = 7^2 \cdot 13 \cdot 17 \cdot 19 \\ \hline \end{array}$$

Figure 1.1: Smooth values of $f(x) = x^2 - 1098413$.

Columns 3, 6, and 7 in Figure 1.2 sum to zero mod 2 and thus the corresponding product

$$\begin{aligned} f(1051)f(1119)f(1142) &= (2^2 \cdot 7 \cdot 13 \cdot 17)\,(2^2 \cdot 7 \cdot 17^2 \cdot 19)\,(7^2 \cdot 13 \cdot 17 \cdot 19) \\ &= 2^4 \cdot 7^4 \cdot 13^2 \cdot 17^4 \cdot 19^2 \end{aligned}$$

gives a square on the right side. Take square roots and recall the definition of f to get

$$(1051 \cdot 1119 \cdot 1142)^2 \equiv (2^2 \cdot 7^2 \cdot 13 \cdot 17^2 \cdot 19)^2 \pmod{n}.$$

This yields $810112^2 \equiv 810112^2 \pmod{n}$, so again we failed to factor n...

	925	1047	1051	1063	1077	1119	1142
$p = -1$	1	1	0	0	0	0	0
$p = 2$	0	0	0	0	0	0	0
$p = 7$	1	0	1	1	1	1	0
$p = 13$	1	0	1	0	1	0	1
$p = 17$	0	0	1	0	0	0	1
$p = 19$	0	1	0	0	0	1	1
$p = 23$	1	0	0	1	0	0	0
$p = 29$	1	1	0	0	0	0	0

Figure 1.2: Binary matrix associated with Figure 1.1.

1.4.2 The third attempt to factor $n = 1098413$

The polynomial $f(x)$ in Section 1.4.1 is not the only serviceable polynomial to produce a congruence mod n with a square on one side. We might pick $g(x) = 841x^2 + 293x - 301$. Note that the discriminant of $g(x)$ is equal to n and that 841 is a square, namely $841 = 29^2$. We therefore have

$$g(x) \equiv \left(29x + \frac{293}{2 \cdot 29}\right)^2 \pmod{n}$$

so that we can easily compute a square root of a value of $g(x)$ mod n. (By $293/(2 \cdot 29)$ above we mean the product of 293 and the inverse of $2 \cdot 29$ modulo n.) We have $g(-1) = 247 = 13 \cdot 19$ and $g(1) = 833 = 7^2 \cdot 17$. Square roots of 247 and 833 modulo n are, e.g., $-29 + 293/58 = -1389/58$ and $29 + 293/58 = 1975/58$ respectively. These can be merged with other data in Figures 1.1 and 1.2 to produce squares on both sides. One such product

$$\begin{aligned}\left(1051 \cdot 1077 \cdot \frac{1975}{58}\right)^2 &\equiv f(1051)f(1077)g(1) = \\ &= 2^4 \cdot 7^4 \cdot 13^4 \cdot 17^2 \pmod{n}\end{aligned}$$

yields $838199^2 \equiv 563108^2 \pmod{n}$. Fortunately $\gcd(838199 - 563108, 1098413) = 1951$, which is a non–trivial factor of n. We have used two polynomials to factor n. In the next section we discuss the use of many polynomials.

1.4.3 Many polynomials

If we evaluate polynomial $f(x)$ of Section 1.4.1 (n is variable now) in points that lie within a distance M of the zero $\sqrt{n}$, then the largest value is about

$2M\sqrt{n}$ (assuming $M \ll \sqrt{n}$). Thus, as with the continued fraction method, the numbers that we like to be smooth, grow as n grows. There is a way to stunt this growth: instead of using one polynomial and evaluating it in many points, we use many polynomials and try fewer points per polynomial to increase the "quality" of the values. This idea originated from Jim Davis and it is an important enhancement of the quadratic sieve. This variant is called the multi–polynomial quadratic sieve, MPQS for short.

The use of many polynomials makes QS much more practical. Davis and Holdridge, who were assigned the task of coding up QS on a Cray supercomputer at Sandia National Laboratories, produced records with MPQS in the 1980s. In 1984 they managed to tackle the number consisting of 71 ones (it has the code R71). *Time* devoted a short article to their success and the *Mathematical Intelligencer* (vol. 6, nr. 3, 1984) had a Cray on the cover plus the decomposition of R71 [39].

Which polynomials do we use? Suppose that we have polynomials $U(x)$, $V(x)$, and $W(x)$ with integer coefficients such that

$$U^2(x) \equiv V^2(x)\,W(x) \pmod{n} \quad \text{for all integers } x \in \mathbf{Z}. \tag{1.3}$$

If $\mathcal{S} \subset \mathbf{Z}$ is a finite subset such that $\prod_{x\in\mathcal{S}} W(x)$ is a square, then we may take

$$X = \prod_{x\in\mathcal{S}} V(x)\sqrt{\prod_{x\in\mathcal{S}} W(x)}, \quad Y = \prod_{x\in\mathcal{S}} U(x)$$

in the modular equation (1.1). So $W(x)$ plays the same role as $f(x)$ in QS. In practice we choose

$$\begin{aligned} U(x) &= a^2x + b, \\ V(x) &= a, \\ W(x) &= a^2x^2 + 2bx + c, \end{aligned}$$

with $|\,x\,| \le M$ (where M is a parameter we choose beforehand) and where a, b, and c are integers satisfying the following conditions [7, p. 117]:

$$a^2 \approx \sqrt{2n}/M, \tag{1.4}$$

$$b^2 - n = a^2c, \tag{1.5}$$

$$|\,b\,| < a^2/2. \tag{1.6}$$

In Section 1.4.5 we describe how a, b, and c may be calculated. The conditions have the effect that the polynomials $W(x)$ are balanced: the absolute values of the minimum and maximum of $W(x)$ on the interval $[-M, M]$ are approximately the same and $W(x)$ is almost symmetrical around the y–axis. See Section 4.2 for computations.

The use of many polynomials enables us to run the quadratic sieve algorithm in parallel on many computers. Each computer is allotted a collection of polynomials in such a way that no machine gets a duplicate polynomial. Output may be written to the local disk.

1.4.4 Precomputing auxiliary numbers

A closer look at Table 1.1 and Figure 1.1 reveals that the prime 3 is missing in all the factorizations. This is not a coincidence: only specific primes occur as prime divisors of $W(x)$. Namely, if a prime p divides $W(x)$, then $p \mid a^2W(x)$ and thus $p \mid (a^2x+b)^2 - n$ (use (1.5)) which means that n is a quadratic residue modulo p. This leads to the definition of the *factor base* $\mathcal{F}$:

$$\begin{aligned}\mathcal{F} \;=\; & \{-1\} \cup \{\text{prime powers } q = p^t \le B \mid \\ & \text{the equation } x^2 \equiv n \pmod{q} \text{ is solvable}\}.\end{aligned}$$

Of course, a prime may divide $W(x)$ more than once, so we also have to account for prime powers. The modular equation $x^2 \equiv n \pmod{q}$, where q is a power of an odd prime p, is solvable if and only if $\left(\frac{n}{p}\right) = 1$. (As usual $\left(\frac{n}{p}\right)$, where p is an odd prime, denotes the Legendre symbol. It is 0 if p is a divisor of n, it is 1 if n is a quadratic residue mod p and -1 if it is a quadratic non–residue. See [11, p. 29] for the computation of Legendre symbols.)

Since -1 plays the same role as a prime in the linear algebra phase, we include it in the factor base. If we use the notation $q \in \mathcal{F}$ in the sequel, we assume however that q is not equal to -1, unless stated otherwise. This avoids the cumbersome "$q \neq -1$" in most cases. Note that $\mathcal{F}$ is independent of the choices of a, b, and c, so we may use the same factor base for every proper choice of a, b, and c. The cardinality of $\mathcal{F}$ is approximately equal to half of the number of primes $\le B$.

To find smooth polynomial values we have to know which values $W(x)$ are divisible by an element q of the factor base. For an element $q \in \mathcal{F}$ the values of x for which q divides $W(x)$ may be found as follows. Compute the solution $t = t_q$ of the congruence equation

$$t^2 \equiv n \pmod{q},\ 0 < t \le q/2$$

(see [42, pp. 212 and 287–288]). This has to be done for every element of the factor base. Now, if $q \mid W(x_0)$, then $q \mid (a^2x_0+b)^2 - n$ (use (1.5)) and thus

$$x_0 \equiv a^{-2}(\pm t_q - b) \pmod{q}, \tag{1.7}$$

provided that $\gcd(a, q) = 1$. This is guaranteed by the choice of a (see Section 1.4.5). For each proper choice of a we compute $a^{-2} \bmod q$ for all $q \in \mathcal{F}$. In Section 1.4.5 we describe how these computations may be done. Furthermore, since $W(x)$ is a quadratic polynomial, $q \mid W(x_0 + mq)$ for all integers m and for no other values of W. Thus we can efficiently calculate the places where an element of $\mathcal{F}$ divides the W–values. This idea originated from Schroeppel.

1.4.5 Efficient determination of proper polynomials

The method described here is due to Peter L. Montgomery; it is slightly more efficient than that of Davis and Holdridge. Choose integers r and k such that $1 < k < r$ (typical choices are, e.g., $r = 30$ and $k = 3$). Generate primes $g_1, g_2, \ldots, g_r$, the so–called *g–primes*, such that

$$\begin{aligned} &\text{(i)} \quad g_i \approx \left(\frac{\sqrt{2n}}{M}\right)^{1/(2k)}, \\ &\text{(ii)} \quad \left(\tfrac{n}{g_i}\right) = 1, \\ &\text{(iii)} \quad \gcd(g_i, q) = 1 \end{aligned}$$

for $i = 1, 2, \ldots, r$ and for all $q \in \mathcal{F}$. Let a be the product of k g–primes:

$$a = g_{i_1} \cdot g_{i_2} \cdots g_{i_k},$$

with $1 \le i_1 < i_2 < \cdots < i_k \le r$. Because of (i), this a satisfies condition (1.4).

Let b_i be a solution of the congruence equation $t^2 \equiv n \pmod{g_i^2}$, $(i = 1, 2, \ldots, r)$. Solve the following system of congruence equations (for any choice of the signs)

$$\begin{aligned} x &\equiv b_{i_1} \pmod{g_{i_1}^2} \\ x &\equiv \pm b_{i_2} \pmod{g_{i_2}^2} \\ &\vdots \\ x &\equiv \pm b_{i_k} \pmod{g_{i_k}^2} \end{aligned} \tag{1.8}$$

by means of the Chinese Remainder Theorem. We fix the sign of b_{i_1} to avoid double work. Let b be the solution of this system of equations. Then it follows that $b^2 \equiv n \pmod{a^2}$ so that condition (1.5) holds with $c = (b^2 - n)/a^2$. If $b \ge a^2/2$, then we replace b by $b - a^2$ to satisfy condition (1.6). Since there are 2^{k-1} possible combinations of signs in (1.8), the number of polynomials that can be calculated with one set of r g–primes and a fixed k is $2^{k-1}\binom{r}{k}$.

If a new a has to be chosen then new numbers x_0 subject to (1.7) have to be computed. Since $a = g_{i_1} g_{i_2} \dots g_{i_k}$, we may use

$$a^{-2} \bmod q = g_{i_1}^{-2} g_{i_2}^{-2} \dots g_{i_k}^{-2} \bmod q.$$

Therefore, with the generation of the g–primes we also precompute and store the numbers $g_i^{-2} \bmod q$, $(i = 1, 2, \dots, r)$, for all the prime powers q in the factor base. In Section 1.4.6 we discuss a method to speed up the switching from one polynomial to the other, provided that a sufficient amount of memory is available on the computers.

1.4.6 The self–initializing variant

We present a method of Alford and Pomerance [1] to quickly pass from one polynomial to another in MPQS. See also the work of Peralta [36]. The method has the advantage that a shorter sieve interval may be used so that the polynomial values are smaller compared with those on a larger interval.

Suppose we just computed the product of k g–primes from the set of r g–primes $\{g_1, g_2, \dots, g_r\}$. Take the first k for convenience: $a = g_1 \cdot g_2 \cdots g_k$. The idea is that as soon as we have computed a proper value of b (then c is determined by (1.5)), we can compute another proper value of b via an iterative process. A similar idea holds for the roots mod q ($q \in \mathcal{F}$) of the polynomials. To be precise, b satisfies the condition $b^2 \equiv n \pmod{a^2}$. Using the Chinese Remainder Theorem we obtain 2^{k-1} useful modular values of b:

$$b \equiv +B_1 \pm B_2 \pm \dots \pm B_k \pmod{a^2},$$

for each combination of the signs, where B_i is defined by

$$B_i = \frac{a^2}{g_i^2} b_i \left\{ \left(\frac{a^2}{g_i^2} \right)^{-1} \pmod{g_i^2} \right\}.$$

(Recall that b_i is a solution of $t^2 \equiv n \pmod{g_i^2}$.) Let

$$b^{(1)} \equiv B_1 + B_2 + \dots + B_k \pmod{a^2}.$$

The other values $b^{(i+1)}$ $(i = 1, 2, \dots, 2^{k-1} - 1)$ may now be computed with the formula

$$b^{(i+1)} = b^{(i)} + 2 \cdot (-1)^{\left(\frac{2i}{2^\nu}+1\right) \cdot \frac{1}{2}} \cdot B_\nu \quad (i = 1, 2, \dots, 2^{k-1} - 1),$$

where ν is defined by the number of primes 2 in $2i$; compare with [1, p. 5 (5.2)]. (Note that ν is dependent on i; we suppress the subscript i however for readability.)

For the modular roots of the polynomials we have a similar correspondence. Write $W_{a,b}(x) = a^2x^2 + 2bx + c$ for the moment and fix an element $q \in \mathcal{F}$. Let x_1 and x_2 be the roots mod q of $W_{a,b^{(i)}}(x)$ and x_1', x_2' the roots mod q of $W_{a,b^{(i+1)}}(x)$. It can easily be checked that the following correspondence between the roots holds:

$$x_j' \equiv x_j - 2 \cdot (-1)^{\left(\frac{2i}{2^\nu}+1\right)\cdot\frac{1}{2}} \cdot B_\nu \, a^{-2} \pmod{q} \quad (j = 1, 2).$$

(Compare with [1, p. 5 (5.3)].)

1.4.7 Sieving

We now discuss a problem that we did not touch yet: how do we recognize smooth numbers, in particular smooth polynomial values? Well, suppose we initialize an array $\mathrm{S}[-M, M]$ by $\mathrm{S}(x) = W(x)$ ($x \in \mathbf{Z}$ and $|\,x\,| \leq M$). Since we know the places x where $W(x)$ is divisible by a prime $p \in \mathcal{F}$ (see Section 1.4.4), we run through the array S and divide the content by p at the corresponding places. If we also account for prime powers, then the places x where the content of S equals 1 exactly correspond to the smooth polynomial values $W(x)$. The time for doing this is unbelievably fast compared with factoring each candidate number to see whether it is smooth.

But we can do it faster. Instead of initializing $\mathrm{S}(x) = W(x)$, we put $\mathrm{S}(x) = \log W(x)$ and subtract $\log p$ at the appropriate places instead of dividing by p there. (Subtraction usually is cheaper than division, especially for large numbers.) Now we have to scan the array for zeros (or rather values close to zero to account for rounding errors) instead of ones.

We can do even better yet. The logarithms of polynomial values stay more or less constant on large intervals. The average of $\log|\,W(x)\,|$ on the interval $[-M, M]$ is approximately $\log\left(\frac{M}{3}\sqrt{n/2}\right)$ (see Section 4.2). This value is suggestively called the *report threshold* (RT). Namely, if we now initialize *each* cell $\mathrm{S}(x)$ to zero and *add* the logarithms of p at the right places for each p in the factor base, then we may scan for cells $\mathrm{S}(x)$ whose content exceeds the report threshold. The corresponding values $W(x)$ are serious candidates to be smooth. Since we have very few candidates, it does not take much time to select the really smooth ones among them. We may use, e.g., trial division to retrieve the prime divisors. The process described here to determine the candidates for smoothness is called *sieving*. The interval $[-M, M]$ is called the *sieve interval*. Array S is the *sieve array*. The sieve part is the most expensive one of the quadratic sieve algorithm. In practice it takes more than 90 % of the total run time of the method.

1.4.8 The small primes variant

Sieving with small primes is more expensive than sieving with large primes because we have to go through the array with many small steps. Besides, small primes do not contribute too much to the contents of the cells of the sieve array. Therefore it is customary not to sieve with small primes and prime powers below some threshold that we denote with QT. In order not to lose values $W(x)$ divisible by such small primes, the report threshold RT is lowered by the amount $\sum_{p^t \leq \mathrm{QT}} \log p$. After the selection of those x for which $\mathrm{S}(x) \geq \mathrm{RT}$, the prime factors of the corresponding $W(x)$ are found by comparing, for all $q \in \mathcal{F}$, x with the two values of x_0 in (1.7) (which are computed and stored after the factor base has been computed). In this way values $W(x)$ divisible by one or more of the small primes by which we have "forgotten" to sieve, are not lost. If QT is suitably chosen, this saves a considerable amount of sieve time. This refinement of the quadratic sieve is known as the *small primes variant.*

1.4.9 The multiplier

Often we find a non–trivial factor of n faster by first multiplying n with a suitable chosen small positive integer m ($\leq$ 100, say) and then factor the product mn rather than n. The number m, called the *multiplier*, has to be chosen in such a way that the factor base belonging to mn contains more primes below some bound than the factor base of n does. Indeed, the smallest of two primes is more likely to be a divisor of an integer than the largest. The *Knuth–Schroeppel function* is a measure for the contribution of prime factors to polynomial values $W(x)$. In Section 4.4 we explain how the contribution of prime factors to polynomial values may be computed. Here we merely give the definition of the Knuth–Schroeppel function f:

$$f(m,n) = -\frac{\log m}{2} + \sum_{p \leq B} g(p, mn) \log p,$$

where m is a positive integer and n the number to be factored. The summation runs over all primes $p \leq B$. The function $g(p, mn)$ is defined as follows.

$$g(2, mn) = \begin{cases} 2 & \text{if} \quad mn \equiv 1 \pmod 8, \\ 0 & \text{otherwise.} \end{cases}$$

For odd primes p we define

$$g(p, mn) = 0 \quad \text{if} \quad \left(\frac{n}{p}\right) \neq 1$$

and if $\left(\frac{n}{p}\right) = 1$, we define

$$g(p, mn) = \begin{cases} \frac{2}{p-1} & \text{if } p \nmid m, \\ \frac{1}{p} & \text{if } p|m. \end{cases}$$

Since it is advantageous to have powers of 2 in the factor base, we want the multiplier m to satisfy $mn \equiv 1 \pmod 8$. (In practice we always choose m in such a way.) We generate a finite list of such numbers m and the multiplier is determined by that particular m for which the Knuth–Schroeppel function attains a maximum. From now on we assume that n contains the multiplier m, so that $n = m \times$ "original number to be factored". Often the use of a multiplier improves the run time of the algorithm by a factor greater than 2. See also [38, p. 391].

1.4.10 One large prime variant

The following idea to improve the multi–polynomial quadratic sieve algorithm is based upon a step in the continued fraction algorithm. This improvement is called the *large prime variant* of MPQS (MPQS1 for short). $W(x)$ is allowed to have a factor $R > B$ that is not composed of primes from the factor base. If the cofactor R (after dividing out all factor base primes in $W(x)$) is less than or equal to B^2, it must be a prime. To restrict the amount of disk space needed for storage of the relations (1.3), we only accept factors $R \leq L$, where L is a parameter we choose beforehand. In practice we choose L in such a way that L/B is a number between 10 and 100. We have to lower the report threshold by $\log L$ to find these values $W(x)$ after sieving.

If we have found two values $W(x)$ with the same R, multiplication of the corresponding relations (1.3) yields a relation of the form (1.3) where $W(x)$ only consists of prime powers $q \in \mathcal{F}$ (and R is moved to $V(x)$).

A relation of the form (1.3), where $W(x)$ only consists of primes $q \in \mathcal{F}$, is called a *complete relation.* If $W(x)$ has one prime factor $R \leq L$ (and the others are in $\mathcal{F}$), then the relation is called a *partial relation.*

1.4.11 Two large primes variant

In the large prime variant of MPQS we allow $W(x)$ in (1.3) to have a prime factor R with $B < R \leq L$. In the two large primes variant of MPQS (MPQS2) we also accept $W(x)$ to have a factor $R \leq L^2$ composed of *two* primes $> B$. In this case we call such a relation a *partial–partial* relation (pp–relation for short). Now the problem of finding combinations

of partial and partial–partial relations that yield a *complete* relation may be formulated as finding cycles in an undirected graph: the vertices are the large primes and two vertices (primes) are connected by an edge if there is a pp–relation in which both primes occur. If we consider 1 as a large prime, then a partial relation is represented by adding 1 as a vertex to the graph and connecting 1 with the large prime in the partial relation. We view this partial relation as a pp–relation where one of the large primes is 1. So an edge in the graph corresponds to a partial or partial–partial relation and a cycle corresponds to a set of relations with the following property: if we multiply these relations, then all the large primes in the product occur to an even power. Hence, for the linear algebra step this set may be viewed as a complete relation. To avoid dependent relations one only has to find the *basic* cycles of the graph. Paton [35] gives an algorithm for finding the basic cycles in a graph. We describe it briefly here.

Let G denote the graph. A spanning tree T for G is generated by "examining" the vertices of G. Take any vertex v of G as the root of T. Then consider each edge e in G that has end point v. If the other end point w of e is a vertex of T, then a fundamental cycle is found; it is e together with the unique path in T that connects v and w. If w is not a member of T, then add e (and thus vertex w) to T. In each case delete e from G. When all edges e with end point v have been considered, proceed recursively. The last vertex added to T plays the role of v. Stop after all vertices are examined. (End of description.)

If R is prime then we require $R < L$ to restrict the total number of relations (L is chosen such that partial relations with $L \leq R < B^2$ do not contribute much to the total number of complete relations). If R is composite, its large prime factors may be found, e.g., by using Shanks' SQUFOF algorithm [42, pp. 191–199]. This algorithm has the advantage that almost all numbers that occur during its execution are in absolute value not larger than $2\sqrt{R}$.

In the sequel we study MPQS1 and MPQS2 (both supplied with the sieve idea). Until the spring of 1996 MPQS2 was the record holder. It was used to factor a 129 digit RSA number [2]. The torch was handed over to the number field sieve when RSA130 was factored with it [12]. This was done in 15 percent of the time it would take to factor it with MPQS2.

1.4.12 Complexity, the cradle of sieving

The purpose of this section is to give a rough indication of the minimum number of bit operations the quadratic sieve requires and to gain some insight into how the sieve idea was born. In fact Richard Schroeppel was

led by complexity arguments to a device that came to be known as the *linear sieve*. It is the forerunner of the quadratic sieve and also its inspiration.

We extract some material from [39] but we remark that the numbers that occur there are random numbers instead of polynomial values. Moreover, the text deals with smooth numbers in which in principle every prime $\leq B$ may occur as a divisor. As we have seen, however, the only prime divisors of the polynomial values are primes p such that n is a quadratic residue mod p. Hence, some dust is swept under the carpet. In Section 4.4 we make things more precise.

The probability that a random positive integer $\leq x$ is y–smooth is approximately $\psi(x,y)/x$, where $\psi(x,y)$ denotes the number of positive y–smooth integers $\leq x$. The expected number of random integers that must be examined to find just one that is y–smooth, is approximately the reciprocal, namely $x/\psi(x,y)$. Let $\pi(y)$ denote the number of primes $\leq y$. Then the expected number of random integers that must be examined is about $\pi(y)x/\psi(x,y)$. Using trial division on each number to be investigated takes about $\pi(y)$ steps. Hence the expected number of steps is approximately $\pi(y)^2x/\psi(x,y)$. Now choose y as a function of x to minimize this expression. It turns out that we have to take y about

$$\exp\left(\frac{1}{2}\sqrt{\log x \log\log x}\right).$$

The minimum value then is approximately

$$\exp\left(2\sqrt{\log x \log\log x}\right).$$

We remark that the computations involved in minimizing the expression are by no means easy [10].

In the quadratic sieve we have to interpret x as the size of the polynomial values, which is of order of magnitude $n^{\frac{1}{2}+\epsilon}$ ($\epsilon > 0$ arbitrarily small). The number y is an estimate for the largest prime in the factor base, which is about B in practice. Thus, factoring n with the quadratic sieve should take about

$$\exp\left(\sqrt{2\log n \log\log n}\right)$$

steps when smooth numbers are recognized using trial division on each number. This is a conjecture, not a theorem.

Using the sieve idea we get the conjectured complexity

$$\exp\left(\sqrt{\log n \log\log n}\right)$$

since we recognize smooth values in about $\log\log y$ steps per each of the $\pi(y)x/\psi(x,y)$ numbers to be examined. The factor $\sqrt{2}$ in the exponent is missing here which means that the sieve idea allows a twofold increase in the length of the numbers that could be factored. Thus sieving improves the quadratic sieve enormously.

1.4.13 Putting algorithms in perspective

Sieving seems to be the fastest device to recognize smooth polynomial values. As long as no substantial improvement of the quadratic sieve is available, its usefulness has come to a limit. The number field sieve is currently the best algorithm to factor general form numbers of more than 105 digits. In Table 1.2 we list the most important factoring methods and the (or a) corresponding suitable range of numbers that may be factored with it.

factoring method	best average performance if
Trial division	$n \leq 10^{10}$ (always try this method first)
SQUFOF	$10^{10} < n \leq 10^{20}$
MPQS1	$10^{20} < n \leq 10^{80}$
MPQS2	$10^{80} < n \leq 10^{105}$
Number field sieve	$n > 10^{105}$
Pollard $(p \pm 1)$–methods	n contains a prime factor $\leq 10^{15}$
Pollard ρ–method	n contains a prime factor between 10^{15} and 10^{25}
Elliptic curve method	n contains a prime factor between 10^{25} and 10^{40}

Table 1.2: Methods and a rough indication of the matching range of numbers that may be factored with it. The run time of the first five methods depends only on the size of n. The run time of the latter three methods depends on the size of the smallest prime factors. See [42] for descriptions of the algorithms.

Chapter 2

Hardware and software

Understanding an algorithm and writing an efficient program for it are different things. In particular the latter one is a tedious job. After having taken care of syntax errors, one finally has a version that "works"... to find out that the code does not produce the desired output. By generating tons of diagnostic output and sweat, one has a version that "really" works but turns out to run slowly. We do not report on these programming stages, but concentrate on ways to optimize the code. We start with some remarks about computer hardware. In particular we observe the architecture of the Cray Y–MP vector computer. This is a super computer that we used frequently to contribute to the Cunningham project (see Appendix). After that we discuss two implementations of the algorithm; one of them is optimized for workstations. Finally, we dedicate some words to parallel computing.

2.1 Some computer characteristics

We focus our attention on conventional computers and ignore the (still) exotic ones like the quantum computer or DNA computer.

Computer power is often measured in units of MIPS (Million Instructions Per Second). By convention a 1 MIPS machine is equivalent to the DEC/VAX 11/780, so 1 MIPS–year is one year on a VAX 11/780 [33].

The processor power is of great importance of course, but in general the accessibility pace of data stored in memory on workstations and personal computers stays behind compared with the growth of processor power. There exist some remedial measures for this problem, one of them being the installation of so–called cache memory. Simply stated, cache is a small and expensive amount of memory that is accessible fast by the processor(s). Physical memory is larger and cheaper but it takes much more time to

fetch data from it. Therefore it makes sense to write programs such that the data used most frequently stays in cache as long as possible. In our case it is a good idea to split up the sieve array in blocks such that each block fits in cache. The elements of the sieve array under consideration are then manipulated fast. Of course, at a certain moment the cache has to be refreshed. In general different computers use different cache refreshment strategies.

The Cray Y–MP and Cray C90 super computers, that we used frequently, do not possess cache or virtual memory, i.e., they have no hierarchical memory structure (if we leave vector registers out of consideration). Bluntly stated, everything goes fast on such machines. Therefore, on such super computers it is *not* a good idea to split up the sieve array in blocks, since the overhead is unnecessary.

Peter L. Montgomery discusses some characteristics of the Cray Y–MP vector computer in [27]. We extract some material from his manuscript to give an insight in the Cray Y–MP architecture, thus explaining, albeit partially, why a vector computer is so fast. The Cray C90 is also a vector computer.

A vector computer can perform, e.g, an element–wise addition and/or multiplication on two vectors of numbers, delivering a result vector element per clock period, whereas a scalar operation takes several clock periods per result. A vector computer performs best when doing a single operation or a sequence of operations repetitively to all elements of one or more input vectors. Ideally, every loop in a vectorized program will resemble

$$\begin{aligned}&\text{FOR } i = 1, n\\ &\quad y_1(i) = f_1(x_1(i), x_2(i), \ldots, s_1, s_2, \ldots)\\ &\quad y_2(i) = f_2(x_1(i), x_2(i), \ldots, s_1, s_2, \ldots)\\ &\qquad\vdots\\ &\text{END FOR,}\end{aligned}$$

where each f_j is an operation without side effects, each x_j and each y_j is an array and where each s_j is a scalar *loop invariant* (an expression that does not change during the loop; in particular a loop invariant must not reference the index variable i). More general array subscripts are allowed, but every array subscript should be a linear function $c_1 \cdot i + c_2$ of the index variable i, where c_1 and c_2 are integer–valued loop invariants. If a subscript is $c_1 \cdot i + c_2$, then the *array stride* is c_1 (assuming all arrays are one–dimensional, with one word per array element). The Cray Y–MP architecture performs best when all array strides are odd or are twice an odd number.

There are eight hardware vector registers, each 64 words long. A loop with heading "FOR $i = 1, n$" is broken into pieces if $n > 64$, a process called

strip mining.

Each Cray Y–MP processor has several vector functional units. Unary operators such as the reciprocal take one input vector and produce one output vector. Binary operators such as element–wise addition and multiplication take two input vectors, or one input vector and one scalar input, producing one output vector.

Chaining allows the movement of elements to continue from one vector operation to another. A *chime* is a sequence of vector operations that can be chained with at most two vector loads and one vector store. The time required to execute a chime is its vector length plus the overheads for vector start–ups. One should try to arrange a computation so as to minimize the number of chimes needed. The same vector functional unit cannot be used twice during a chime. During a chime, a vector instruction may reference the results of previously initiated vector instructions, but no vector instruction is allowed to write back to a vector register previously used as input during that chime.

2.2 Implementation

For our MPQS1–experiments we used the implementation described in [40]. Almost all our subroutines are written in Fortran. We have originally implemented the MPQS2–algorithm on a supercomputer like the Cray Y–MP vector computer. We used the same implementation on Silicon Graphics (SGI) workstations but it turned out that especially the sieve part could be improved significantly by using blocking strategies on workstations with a reasonable amount of primary and secondary cache memory (see Section 2.2.2).

2.2.1 Implementation without blocking strategies

The sieve operations (i.e., addition of $\log p$ to an element of the sieve array) are done in 64–bits floating–point arithmetic on the Cray Y–MP and in 32–bits floating point arithmetic on SGI workstations. The maximum speed we obtained (in millions of sieve operations per second) was 3.3 on a 100 MHz Silicon Graphics workstation, 110 on the Cray Y–MP [41] and 270 on the Cray C90. The maximum speed was 5.7 on SGI when we used blocking strategies. We used a package of D. T. Winter of CWI to carry out multi–precision integer arithmetic. For the large prime R occurring in the partial relations we accepted R with $B < R < L$ and those with $L \leq R < B^2$ were rejected. We implemented Paton's cycle finding algorithm [35] and used it

as a preprocessing step for the Gaussian elimination step in MPQS2. An algorithm for just *counting* (but not finding) the basic cycles ([25, pp. 789–790] and [15, pp. 61–64]) was implemented by us as a tool to check during the sieve part of MPQS2 whether sufficiently many relations (complete, partial, and partial–partial) were collected. The method used to do the Gaussian elimination (mod 2) is described in [34]. The elements of the bit–array are packed in words of 64 bits (on the Cray computers) or 32 bits (on the Silicon Graphics). This allows the use of XOR–ing (exclusive or) with the column vectors of the array, which is very efficient. The total Gaussian elimination step (including finding basic cycles) takes less than 0.6 % of the total work of the MPQS2–algorithm.

2.2.2 Implementation with blocking strategies

We discuss the "blocked" sieve part of the algorithm and show some other cunning contrivances.

The straightforward implementation of the heart of the sieve part (adding of logarithms to the entries – or cells – of the sieve array) resembles the pseudo code in Figure 2.1.

```
1. initialize the sieve array S(−M, M)
2. FOR prime powers q = p^t ∈ F, q > QT
3.    FOR x ∈ [−M, M] with x ≡ x_1  (mod q) or x ≡ x_2  (mod q)
4.       S(x) = S(x) + log p
5.    END FOR
6. END FOR
```

Figure 2.1: Naive implementation of the sieve part.

Here x_1 and x_2 are the modular roots of the polynomial under consideration. The meaning of the other symbols is evident. When the implementation of Figure 2.1 runs on a conventional workstation, the most time consuming part is the access of the elements $\mathrm{S}(x)$ in line 4 since there is a cache miss in most of the cases.

We speed up the implementation by sieving with a small part of the sieve array that just fits in cache. After selecting the candidates for smoothness and saving the relations found, we pass to the next part of the sieve array until the whole array has been processed.

We can do even better by collecting a number of entries of the sieve interval into one machine word, thus keeping more entries in cache. To be precise, we implemented the algorithm on a 32 bits machine where each byte

(non–addressable when using Fortran) consists of 8 bits. For the numbers we have factored, the sum of the logarithms never exceeds $2^8 - 1$ so that the sum fits in one byte. Hence, one word can contain four entries of the sieve interval. Assume, for convenience, that the length A of the part of the sieve array under consideration is divisible by 4. We declare an array $S'(1, \frac{A}{4})$ that represents the block of length A; S′ contains A entries of the sieve interval. Thus, by packing four entries into one word, the block that stays in cache contains four times as many entries as in the non–packed case.

There is another advantage to the pack method but first we have to explain what a *mask* is used for. A mask is nothing more than an integer constant. It is used to "examine" other words. Suppose, for example, that the first five bits of the mask M (starting from the right with bit position zero) are equal to 0 and the other bits on the left are equal to 1. Now we can use mask M to scan for words W that are larger than $2^0 + 2^1 + 2^2 + 2^3 + 2^4 = 2^5 - 1$: perform a logical AND operation on M and a word W to be examined. This means that the corresponding bits of M and W form the argument of a logical AND operation and the resulting value (0 or 1) is put on the same position in the resulting word R. If R is zero, we know that W does not contain bits equal to 1 left from bit position four and hence W is smaller than 2^5. The bits of M may be considered as shutters: a shutter is open if the bit is 1 and closed if it is 0.

To select candidates for smoothness we have to scan the sieve array for entries containing a byte whose content exceeds the report threshold. We do a rough sift by using a large mask on each entry of S′. By AND–ing the mask with an entry we can see if the entry contains a byte whose content exceeds the report threshold. If the result is zero, then the entry does not contain candidates. If it is non–zero, then at least one of the bytes is a candidate. In this way we check 4 bytes (each representing a place in the sieve interval) at once. To determine the precise byte position of a candidate in a word, we use smaller masks. Since in most cases the result is zero, the speed of selecting the candidates is about four times the speed of selecting candidates by examining entries that represent only one place in the sieve interval.

If the report threshold lies, e.g., in the interval $[32, 64)$ then the large mask has the value $(2^{31} + 2^{30} + 2^{29}) + (2^{23} + 2^{22} + 2^{21}) + (2^{15} + 2^{14} + 2^{13}) + (2^7 + 2^6 + 2^5)$. Thus, the three left most bits of each byte are set to 1 and the first 5 bits on the right of each byte are 0. Since on our workstations the left most bit controls the sign of an integer, the mask is represented by the integer value -1625350368 in this case. The numbers between parenthesis in the summation above are the small masks.

There is a price to pay for the pack strategy: the logarithm of a prime of the factor base must be shifted to the left before the logarithm can be added to a byte. Nevertheless, the cache effect dominates and the strategy improves the speed of the program. An experiment illustrates the effect. We factored a 50–digit number with a blocked and unblocked implementation variant of MPQS1. The sample number is

C50 4993267058 9812986150 1743741922 0841046002 3163760841.

In the sequel Cx means a composite number with x decimal digits. The parameters chosen are

$$\begin{aligned} B &= 6 \times 10^4 \\ \mathrm{QT} &= 30 \\ L &= 10^7 \\ M &= 5 \times 10^5 \\ A &= 2.5 \times 10^5 \quad \text{(blocked)} \end{aligned}$$

The resulting report threshold RT is 53 (see Section 2.2.3 for the practical computation of RT). We factored the C50 on an SGI workstation with one 100 MHz R4000 processor, primary data and instruction cache size of 8 Kb, and secondary unified instruction/data cache size of 1 Mb. We blocked it with respect to the secondary cache. With the unblocked variant we factored the C50 in about 1047 seconds while the blocked variant managed it in about 55 % of that time, 575 seconds to be exact. See also [46].

2.2.3 Some gadgets

The excellent book of Henri Cohen [11] contains many algorithms and tricks. We eagerly made use of it.

We check whether the number n to be factored has prime factors smaller than B and if so, we remove them and substitute the cofactor for n. Then n is tested on being prime with the Rabin–Miller test [11, p. 415]. In practice the numbers to be done by MPQS often endured a factoring algorithm like the elliptic curve method (ECM) [11, pp. 476–482] that finds factors of less than 35 digits.

Besides the parameters to be chosen beforehand, the input file to be read by the program contains numbers that control with how much the "original" report threshold $\log\left(\frac{M}{3}\sqrt{n/2}\right)$ must be lowered. Indeed, we must lower it theoretically with an amount of $\sum_{p^t \le \mathrm{QT}} \log p + \log L^2$ to rescue relations that would have been missed due to the small and large primes variants and rounding errors. In practice we lower it with an amount of

$a_1 \cdot \sum_{p^t \leq QT} \log p + a_2 \cdot \log L^2$, where a_1 and a_2 are often chosen to be 0.05 and 0.5 respectively. Experiments learn that we get more relations per time unit in this way. Playing a bit with the value of RT shows that the optimal value of RT fluctuates around the "theoretical" value [23].

The determination of the greatest common divisor of two integers can be implemented straightforward if Euclid's algorithm is used. We coded up the algorithm described in [11, pp. 14–15] however, since it is faster on binary machines. The most important operations done in the binary gcd algorithm are subtractions and divisions by 2 that are just integer shifts. The computation of the modular inverse (if it exists) of an integer is also easy to implement, but we used the binary version displayed in [11, p. 16]. The same holds for the computation of the Legendre symbol [11, p. 29]. The square root (if it exists) mod p (p an odd prime) is determined following [11, p. 33]. Extracting square roots modulo prime powers with exponent > 1 can be done using the roots modulo the powers that have exponent one less. Repeated squaring ("binary method") is a well–known trick for modular powering [11, pp. 8–12]. We used the implementation in [42, pp. 195–197] of Shanks' Square Forms Factorization Method (SQUFOF) to factor the composite part of polynomial values whose two largest prime factors are not a member of the factor base.

We built in a signal handler in our program that catches termination or kill signals to give the program some moments to write data to a recovery file before it terminates. With this device we are able to start up the program again without doing duplicate computations: the program first scans for an existing recovery file and reads in relations possibly found. It then continues where it was interrupted previously.

The implementation of the blocked self–initializing variant of MPQS2 consists of 45 modules and contains about 8000 lines in total.

2.2.4 Parallel computing

We factored many numbers (with MPQS2) to update the table of Brent and te Riele [6], see the Appendix. Many numbers were done with the Cray Y–MP and Cray C90 vector computers installed at Stichting Academisch Rekencentrum Amsterdam (SARA). When we arrived at numbers with more than 90 digits, even the super computer took too much time *and* money to factor them. The CPU time allotment namely was arranged in such a way that users with a shorter job to do were pushed forward in the waiting list. (CPU = Central Processing Unit) Besides, every user had to pay an amount of money for each time unit consumed. Thus factoring integers became literally expensive. Also the notion that 30 modern SGI workstations together

have the same power roughly as the Cray Y–MP, prompted us to compute on workstations that are free to use instead of on a super computer.

In the spring of 1995 we started sieving in parallel at the Department of Mathematics and Computer Science of Leiden University. After requesting the "owners" of 86 workstations for using their machines during the nights and weekends, about 80 % consented. That left us with 69 machines to run our self–initializing blocked implementation of MPQS2. Each machine was assigned a collection of polynomials in such a way that duplicate computations could not occur. We installed programs, developed at CWI, that controlled automatic start–up and termination of the sieve program. The linear algebra part was always done on a fast and big SGI Power Challenge at CWI.

Chapter 3

Comparison of two variants

3.1 Introduction

When we try to factor integers with a given number of digits with one of the variants of QS, then the run time does not fluctuate much if we keep the parameters and the computer system fixed. We present a formula that predicts the run time of MPQS2 in this case. Different variants however give rise to different run times. We compare the one large prime variant of QS (MPQS1) with the two large primes variant (MPQS2).

MPQS2 is known to be faster than MPQS1 "by approximately a factor of 2.5 for sufficiently large n" [25], but the cross–over point depends heavily on the choice of the parameters in the two methods, the computer, the available memory, and the implementation. It is stated further in [25] that MPQS2 was found to be faster than MPQS1 for numbers of at least 75 decimal digits, and that the speed–up factor of 2.5 was obtained for numbers of more than 90 digits. As a comparison, a 106–digit number was factored with MPQS1 in about 140 MIPS years, and a 107–digit number with MPQS2 in about 60 MIPS years, both with a factor base size of 65500. A 116–digit number was factored with MPQS2 in about 400 MIPS years, with a factor base size of 120000. No actual results for smaller numbers were given. In Thomas Denny's Master's Thesis [15] various experiments with MPQS2 are reported for numbers in the 75–95 decimal digits range. From these experiments it is not clear where the cross–over point for Denny's implementation lies. We experimented to search for the cross–over point for our implementation. The experiments and conclusions are recorded in Section 3.2.

The largest numbers presently factored with MPQS2 are a 120–digit number done in about 825 MIPS years [16], and the 129–digit RSA challenge described by Martin Gardner, done in about 5000 MIPS years with a factor base size of 524339 [2].

In Section 3.3 we present a way to predict the sieve time when many numbers of about the same size are factored with fixed parameters.

3.2 Experiments

To compare MPQS1 with MPQS2 we have run our (unblocked) implementations of these algorithms on the Cray C90 for the 71–digit number

$$\text{C71} = (10^{71} - 1)/9$$

and for the 87–digit cofactor

$$\begin{aligned}\text{C87} \;=\; & 1360245\ 9257583786\ 3939661047\ 9463908049\ 3042354284\backslash \\ & 1197990430\ 2204441489\ 2390146207\ 9070640121\end{aligned}$$

of $72^{99} + 1$. For C71, four experiments with different combinations of B, L/B, and M were carried out where in the second, third and fourth experiment only one of the three parameters was changed compared with the previous experiment. The value of QT was kept fixed on 40.
The number

$$\begin{aligned}\text{C80} \;&=\; \frac{75^{64}+1}{2 \times 224914177 \times 151\ 1139087864\ 2191703680\ 6943723393} \\ &=\; 1484463729\ 7924826822\ 3924402812\ 7205475762\backslash \\ &\quad\ 2335589237\ 4279886592\ 8124925295\ 6234072833 \qquad (3.1)\end{aligned}$$

(having the two prime factors 68 7990387865 1231938882 1350925569 and 21576809 1527974049 6462476159 5710136567 7594246657)

was factored with MPQS2 (see Section 3.3) and we made a comparison run with MPQS1 for $B = 10^5$, $M = 3 \times 10^6$, QT = 50, and $L/B = 400$ (the optimal choice for MPQS2 on the Cray C90). The results are given in Tables 3.1 and 3.2.

For C71, the parameter choice $B = 3 \times 10^5$, $L/B = 20$, and $M = 5 \times 10^5$ yields a somewhat smaller sieve time for MPQS2 (0.55 CPU hours) than for MPQS1 (0.58), but if we allow more memory use by choosing $B = 6 \times 10^5$ and $M = 2.5 \times 10^6$ (and $L/B = 40$), then MPQS1 beats MPQS2 (0.29 vs. 1.21). Increasing the length of the sieve interval (M from 5×10^5 to 2.5×10^6) particularly improves the efficiency of MPQS1 (and, to a lesser extent, of MPQS2). For C87, with the parameter choice $B = 5 \times 10^5$, $L/B = 20$, and $M = 2.5 \times 10^6$, MPQS2 is faster than MPQS1 (11.9 vs. 16.4).

We conclude that for our implementations MPQS2 beats MPQS1 for numbers of more than 80 decimal digits, but the cross–over point strongly depends on the amount of available central memory. For practical reasons (like throughput) it may be profitable to reduce the size of a sieve job on the Cray C90, so even though such a computer has a large central memory, it is worth while to restrict the size of the upper bound on the primes in the factor base and to have an efficient implementation of a memory–economic method like MPQS2. This aspect is even more important on workstations, particularly when there are primary and secondary cache memories (as is usual on workstations).

					MPQS1			
	$\frac{B}{10^5}$	n_f	L/B	M	T_s(h)	n_c	n_1	$n_{c,1}$
C71	3	12979	20	5.0×10^5	0.58	10204	17993	2784
C71	6	24510	20	5.0×10^5	0.56	20827	23794	3703
C71	6	24510	40	5.0×10^5	0.55	20312	30399	4209
C71	6	24510	40	2.5×10^6	0.29	20196	31034	4359
C80	1	4806	400	3.0×10^6	13.4	1580	49143	3229
C87	5	20838	20	2.5×10^6	16.4	9902	70029	10940

Table 3.1: Comparison of MPQS1 with MPQS2 for C71, C80, and C87. n_f = number of elements in the factor base; T_s = sieve time; n_c = number of complete relations found immediately; n_1 = number of partial relations found; $n_{c,1}$ = number of complete relations from the partial relations.

	MPQS2					
	T_s(h)	n_c	n_1	$n_{c,1}$	n_2	$n_{c,2}$
C71	0.55	5063	36468	4709	42617	3400
C71	0.96	10868	68019	8383	70395	5389
C71	1.28	9817	80017	7390	132290	7412
C71	1.21	9803	81612	7499	138147	7969
C80	5.67	618	91332	634	193278	3598
C87	11.9	7009	63089	8220	57513	5620

Table 3.2: Continuation of Table 3.1. n_2 = number of pp–relations found; $n_{c,2}$ = number of complete relations generated by combining the partial relations (with different large primes) and the n_2 pp–relations.

Furthermore, with our MPQS1–program we have factored the 99–digit

cofactor

168483084 9783397621 1530436039 97266025308430041776\
9257490404 3633682183 8963842217 5595211200 8347771913

of the *more wanted number* from the Cunningham Table with code "2,914M C133". This "C133" is the composite number of 133 decimal digits $(2^{457} + 2^{229} + 1)/(5 \times 71293)$; Peter L. Montgomery had found the 34–digit prime factor

6196 3339792346 7946602186 4314534473

of this number with the elliptic curve method and left the 99–digit composite cofactor. We decomposed the C99 into the product of the 49– and 50–digit primes:

584529625 7595668545 5249699376 9750792368 2374822769

and

2882370329 1241135239 3780756160 7800380643 3692452377,

with the help of an eight processor IBM 9076 SP1 and 69 Silicon Graphics workstations (63 at CWI and 6 at Leiden University). The factor base size was 56976 with $B = 1.5 \times 10^6$, $L/B = 50$, $M = 2 \times 10^6$, and QT = 30. Parallel processing with good load balancing was effectuated by assigning different polynomials to different workstations. The total amount of sieve time was about 19500 workstation CPU–hours. The physical time for this factorization was about four weeks. This means that we consumed about 40 % of the total CPU–capacity of these workstations during that period (assuming that they all are equally fast: in fact, an RS 6000 processor of the IBM SP1 sieved about twice as fast as an SGI workstation). The Gaussian elimination step was carried out on a Cray C90; it required about 0.5 Gbytes of central memory and one hour of CPU–time.

As a comparison with a vector computer [41], on a Cray Y–MP we factored a 101–digit more wanted Cunningham number with MPQS1 in 475 CPU–hours, using $B = 1.3 \times 10^6$, with 50179 primes in the factor base, $L/B = 50$, $M = 4.5 \times 10^6$, and QT = 40 (our MPQS1–implementation runs about twice as fast on the Cray C90 as on the Cray Y–MP).

As a comparison with MPQS2, from the MPQS2 results listed in Table 3.2 we estimate (based on the rule of thumb that the computing time of MPQS2 approximately doubles if the size of the number increases by three decimal digits) that we would roughly need 10000 CPU–hours of an SGI workstation to factor the 99–digit cofactor of 2,914M C133, yielding a speed–up factor

of about 2 compared with MPQS1.
If we take a factor 1.64 (see the next paragraph) instead of 2, then the time is less than 4000 CPU–hours.

In the Appendix (Tables A.1 – A.2) we list the results of our experiments with MPQS2 on eight numbers in the 66–83 digit range on an SGI workstation and 73 numbers in the 67–88 digit range on a Cray C90 vector computer. Most of these numbers fill gaps in the table [6] and are difficult to factor (they were tried before with ECM without success). We have varied the different parameters B, L/B, and M on different numbers (but not in a very systematic way) and kept QT = 40 fixed. We observe that the average CPU–time for numbers in the 67–88 digit range varies between 0.4 and 12 CPU–hours, so that increasing the number of digits by three gives an increase of the sieve time by a factor of about 1.64. This is smaller than the factor of 2 that is usually observed for MPQS1.

3.3 Analysis of the sieve time of MPQS2

A theoretical and practical problem with MPQS2 is the determination of the optimal parameters for a number of a given size. Since it only pays to use MPQS2 for large numbers, and since it is difficult to accurately predict the total running time of MPQS2 on the basis of a short test run (as contrasted with MPQS1), the precise effect of one specific choice of the parameters can only be measured accurately by carrying out the complete sieve part of the job. So, to find the *optimal* parameter choice that minimizes the CPU–time for a given number, one has to *repeat* the complete sieve job for several (10, say) different choices of the parameters. Of course, this does not make much sense since *one* sieve job factors the number. So we decided to adopt the strategy to factor as many as possible *different* numbers in a not too wide decimal digits range, thus providing extensive experience with MPQS2 for many different numbers on the one hand and contributing to a table of unfactored numbers [6] on the other hand. The price to pay for this strategy is that we can only give an *indication* of the optimal parameter choice for MPQS2 for numbers in the 65–90 decimal digits range.

We would like to estimate the time that MPQS2 spends on the sieve step for numbers n of about d decimal digits, given B, M, L, and QT. Let

$$\begin{aligned} f_1 &= n_c/n_f, \\ f_2 &= n_2/n_1. \end{aligned}$$

(See the list of symbols for notations.) During the sieve step, the numbers n_c, n_1 and n_2 grow (more or less) linearly with the time, so that also the

#	33	35	37	44	42	38	48	40
m	109	37	1	109	1	109	5	29
f_1	0.243	0.244	0.255	0.269	0.275	0.297	0.301	0.310
f_2	5.98	5.79	4.04	3.68	2.75	2.37	2.13	2.29
#	47	34	39	36	46	41	32	43
m	1	1	1	7	43	1	1	41
f_1	0.320	0.325	0.331	0.346	0.348	0.349	0.352	0.363
f_2	1.70	1.64	1.14	0.906	0.961	0.862	0.760	0.798

Table 3.3: Values of f_1 and f_2 measured for 16 numbers n from Table A.2 (identified by the number in the first row). We used $d = 86$, $B = 5 \times 10^5$, $M = 1.5 \times 10^6$, $L/B = 20$, and QT $= 40$. For each n we computed a multiplier m and factored mn instead of n.

fraction f_1 grows linearly, and f_2 remains more or less constant (after the sieve step has been running for a short time). We observed that the values of the fractions f_1 and f_2, measured after completion of the sieve step, seem to be related. For example, Table 3.3 gives the values of f_1 and f_2 measured for 16 numbers factored on the Cray C90 with $d = 86$, $B = 5\times10^5$, $M = 1.5 \times 10^6$, $L/B = 20$, and QT $= 40$. For each of the 16 numbers n we computed a multiplier m and factored mn instead of n. Table A.2 in the Appendix contains more information about the 16 numbers.
Table 3.3 suggests that f_2 is an exponential function of f_1:

$$f_2 = ae^{bf_1}$$

for some constants a and b. Based on the table, we estimated $a = 315$ and $b = -16.5$. Since $\log f_2 = \log a + bf_1$, it follows that $n_c = \frac{1}{b}(\log f_2 - \log a)\cdot n_f$. If u is the time needed to generate one complete relation, we obtain the following approximation for the sieve time $T_s = un_c$:

$$T_s \approx (0.349 - 0.061 \log f_2) \cdot u \cdot n_f. \tag{3.2}$$

We estimate u and f_2 by letting the program run for a short while, five minutes say. The measurements shown in Table 3.4, pertaining to runs on the Cray C90 of several 85– and 86–digit numbers, suggest that the estimate works well. (The test numbers are composite factors of the numbers in the column below "n"; 98 91+ means $98^{91} + 1$, 47 67 – means $47^{67} - 1$; the multiplier is denoted by m).
Consequently, the approximation (3.2) may be used to obtain a good estimate of T_s in the MPQS2–algorithm for numbers of about the same size, and fixed parameters B, M, L, and QT. For numbers in another range, or

#	m	u	f_2	n_f	(3.2)	T_s
21	19	5.140 s	1.1945	20741	10.0 h	9.8 h
22	1	4.518 s	0.7646	20744	9.50 h	9.8 h
24	1	3.357 s	1.4378	20930	6.37 h	6.0 h
31	1	4.226 s	1.0866	24641	9.94 h	10.0 h
48	5	8.785 s	2.1364	20911	15.4 h	15.4 h

Table 3.4: Tests of approximation 3.2. For five composite numbers from Table A.2 (identified by the number in the first column), we measured the actual value of T_s and computed the value predicted by the approximation (one but last column).

if we wish to change the parameters, some experiments have to be done to determine the total sieve time under these new conditions, by which a and b, whence the coefficients in (3.2), can be estimated.

To test the dependence of T_s on L, we carried out the *complete* sieve step of MPQS2 for the 80–digit number 3.1 on the Cray C90, with $B = 10^5$, $M = 3 \times 10^6$, QT = 50 fixed, and for various different values of L. The statistics are shown in Table 3.5.

In the partial relations we accepted the large prime R to be $< B^2$. (We get these relations free because $R < B^2$ implies that R is prime.) For $B = 10^5$ the number of elements in the factor base is 4806. The sieving was continued until the total number of complete relations, including those generated by the partial relations and the partial–partial relations, surpassed this number. We only measure the total number of complete relations obtained so far at selected points in our program, so the *actual* total number of complete relations is usually somewhat larger than the number of elements in the factor base.

As we increase L/B, the program generates more partial–partial relations and less complete and less partial relations in a fixed amount of sieve time. For $L/B \leq 400$ the gain in complete relations ($n_{c,2}$) generated by the pp–relations (n_2) more than sufficiently compensates for the loss of complete relations directly found by the sieve (n_c) and the loss of complete relations ($n_{c,1}$) generated by the partial relations (n_1). As a result, the total sieve time T_s *goes down*. For $L/B > 1000$, however, the increase in size of the large primes in the partial and partial–partial relations is responsible for a decrease in the number of complete relations derived from these relations, and also the time that SQUFOF needs to find the two large primes in a pp–relation increases, so now the resulting total sieve time *increases*. Consequently, the minimal sieve time on the Cray C90 is reached if we choose

L/B	T_s	n_c	n_1	$n_{c,1}$	n_2	$n_{c,2}$	total
30	8.64 h	1036	129318	1661	29143	2121	4818
60	7.06 h	871	117532	1249	51929	2739	4859
100	6.49 h	775	109506	1025	76324	3070	4870
200	6.02 h	685	99474	795	123001	3339	4819
400	5.67 h	618	91332	634	193278	3598	4850
600	5.71 h	578	87265	568	243015	3698	4844
800	5.62 h	563	84926	531	291177	3766	4869
1000	5.75 h	546	83082	501	333726	3796	4843
1600	6.19 h	521	79960	464	445526	3860	4845

Table 3.5: Number of relations as a function of L, for the factorization of (3.1) with $B = 10^5$, $M = 3 \times 10^6$, and QT $= 50$. The column $n_{c,1}$ is the number of complete relations generated by the n_1 partial relations, and $n_{c,2}$ is the number of complete relations generated by combining the partial relations (with different large primes) and the n_2 pp–relations. "Total" is the sum $n_c + n_{c,1} + n_{c,2}$.

L/B in the interval $400 \leq L/B \leq 1000$. In that interval the total sieve time is only slightly varying. We conclude that, in order also to minimize the amount of *memory* for storage of the relations, the optimal choice of L/B is about 400.

Chapter 4

The number of relations

4.1 Introduction

We consider the one large prime variant (MPQS1) of the algorithm. If we can predict the rate by which the complete relations in MPQS1 are generated as a function of the various parameters in the algorithm, then we can determine a good choice of the parameter values. Here we give a method to do so.

Section 4.2 contains notation and preparation. Counting and approximating the number of smooth integers in an interval are the subjects of Section 4.3. Section 4.6 contains a method to predict the number of complete relations that descend from the incomplete relations. In Sections 4.4 and 4.5 we give an approximation of the numbers of complete and incomplete relations per polynomial in MPQS1. We present numerical results in Section 4.7. We analyze the total amount of work in Section 4.8 and draw conclusions in Section 4.9.

4.2 Notation and preparation

We write $\log x/\log y$ for $(\log x)/\log y$. In the sequel u denotes $\log x/\log y$. Euler's constant is denoted by γ $(= 0.5772...)$.

Recall that the number of y–smooth positive integers $\leq x$ is denoted by $\psi(x, y)$. We change the notation of the report threshold and the polynomial $W(x)$ for convenience:

$$\begin{aligned} T &: \text{sieve threshold,} \\ W(x) &= ax^2 - 2bx - c : \text{sieve polynomial,} \\ \Gamma &: \text{graph of } W. \end{aligned}$$

We assume that the number n to be factored is composite, already contains the multiplier m, does not contain prime divisors $\leq B$ (except possibly for the primes in the multiplier), and that n is not a perfect power. The multiplier m is chosen such that $n \equiv 1 \pmod 8$. Hence 2 is a member of the factor base. In practice (when using a workstation) we choose B between 10^5 and 10^6 and L between $10B$ and $100B$. If n has about 100 decimal digits, then M is about 10^7. We say that an integer is a W–value if it equals $W(x)$ for some integer x in the sieve interval. The integers a, b, and c satisfy the following conditions:

$$a \approx \sqrt{2n}/M, \quad |b| < a/2, \quad b^2 + ac = n.$$

Let $-R$ be the minimum and S the maximum of W on the sieve interval. The minimum of W is attained at $x = b/a$ and the maximum at the boundary of the sieve interval. We have $aW(x) = (ax - b)^2 - n$ so that $aR = -aW(b/a) = n$. Furthermore, $aS = aW(\pm M) = (\pm aM - b)^2 - n \approx (\pm\sqrt{2n} - b)^2 - n \approx n$ since $|b| < a/2 < \sqrt{n}/M$ and M is large. Thus $R \approx S \approx n/a \approx M\sqrt{n/2}$. Note that also $c \approx S$. Theoretically $T = \log\left(\frac{1}{3}M\sqrt{n/2}\right) - \log L$, but in practice we have to lower this value a bit to get more relations per time unit.

We use the term complete relation for a B–smooth W–value. An incomplete relation is a W–value y that is divisible by a large prime q, $B < q \leq L$, such that y/q is B–smooth. Let t_1 and t_2 be the number of complete and incomplete relations respectively.

There are relatively few W–values in the interval $(-e^T, e^T)$ compared with the total number $2M + 1$ of W–values. Indeed, if $W(x) = y$ and $x \geq b/a$, then $x = \frac{b}{a} + \frac{1}{a}\sqrt{n + ay}$ so that the number of W–values in $(-e^T, e^T)$ is approximately

$$\frac{2}{a}\left(\sqrt{n + ae^T} - \sqrt{n - ae^T}\right) = \frac{4e^T}{\sqrt{n + ae^T} + \sqrt{n - ae^T}}.$$

If $T \approx \log\left(\frac{1}{3}M\sqrt{n/2}\right) - \log L$ and $a \approx \sqrt{2n}/M$, then the number of W–values in $(-e^T, e^T)$ is approximately

$$\frac{2\sqrt{2}M/(3L)}{\sqrt{1 + 1/(3L)} + \sqrt{1 - 1/(3L)}} \approx \frac{\sqrt{2}}{3L}M$$

and this is only a small fraction of M.

The set

$$\Gamma_1 = \{(x, y) \in \mathbf{R}^2 \mid x > b/a,\ y = W(x),\ e^T \leq y \leq S\}$$

is called the right upper branch of Γ on the sieve interval. The right lower branch of Γ on the sieve interval is the set

$$\Gamma_2 = \{(x, y) \in \mathbf{R}^2 \mid x > b/a,\ y = W(x),\ -R \le y \le -e^T\}.$$

The left upper branch Γ_3 and left lower branch Γ_4 of Γ on the sieve interval are defined similarly: replace $x > b/a$ by $x < b/a$ in the corresponding definitions of the right upper and lower branch.

Let $t_{1,i}$ and $t_{2,i}$ be the total number of complete and incomplete relations of Γ_i $(i = 1, 2, 3, 4)$ respectively. We have $t_1 \approx \sum_i t_{1,i}$ and $t_2 \approx \sum_i t_{2,i}$ since there are relatively few W–values in the interval $(-e^T, e^T)$.

4.3 Smooth integers in an interval

The Dickman–De Bruijn function ρ plays a key role in approximating the number of smooth integers below some bound. The function is defined by the differential–difference equation

$$\begin{aligned} \rho(u) &= 1 \quad (0 \le u \le 1), \\ u\rho'(u) + \rho(u-1) &= 0 \quad (u > 1). \end{aligned}$$

We have

$$\rho(u) = 1 - \log u \quad (1 \le u \le 2). \tag{4.1}$$

From the definition of ρ it follows that ρ is piecewise analytic and that ρ agrees with an analytic function ρ_d on the interval $[d-1, d]$ $(d = 1, 2, \ldots)$. We expand the Taylor series of ρ_d in a left neighborhood of $u = d$:

$$\rho(d - \xi) = \rho_d(d - \xi) = \sum_{i=0}^{\infty} c_i^{(d)} \xi^i \quad (0 \le \xi \le 1).$$

Bach and Peralta [3] describe an efficient method due to Patterson and Rumsey to compute the coefficients $c_i^{(d)}$ iteratively. Since $\rho_1(u) = 1$ $(0 \le u \le 1)$, we have $c_0^{(1)} = 1$ and $c_i^{(1)} = 0$ for $i > 0$. The coefficients $c_i^{(2)}$ may be computed from (4.1):

$$\rho(2 - \xi) = 1 - \log 2 - \log(1 - \xi/2) = 1 - \log 2 + \sum_{i=1}^{\infty} \frac{\xi^i}{i\, 2^i} \quad (0 \le \xi \le 1).$$

In general we have

$$c_i^{(d)} = \sum_{j=0}^{i-1} \frac{c_j^{d-1}}{i\, d^{i-j}} \quad \text{for } i > 0 \quad \text{and} \quad c_0^{(d)} = \frac{1}{d-1} \sum_{j=1}^{\infty} \frac{c_j^{(d)}}{j+1}.$$

Empirically, Bach and Peralta found that 55 coefficients are sufficient to compute ρ to IEEE standard double precision (with a relative error of about 10^{-17}) in the range $0 \leq u \leq 20$. In Table 4.1 we list rounded values of $\rho(d)$ for integers d in the range $2 \leq d \leq 11$. These values were computed from the Taylor series of ρ_d in a left neighborhood of $u = d$ using the first 55 coefficients. See also [26].

d	$\rho(d)$	d	$\rho(d)$
2	0.306853	7	$0.874567 \cdot 10^{-6}$
3	$0.486084 \cdot 10^{-1}$	8	$0.323207 \cdot 10^{-7}$
4	$0.491093 \cdot 10^{-2}$	9	$0.101625 \cdot 10^{-8}$
5	$0.354725 \cdot 10^{-3}$	10	$0.277017 \cdot 10^{-10}$
6	$0.196497 \cdot 10^{-4}$	11	$0.664481 \cdot 10^{-12}$

Table 4.1: Rounded values of the Dickman–De Bruijn function ρ.

For the number $\psi(x, y)$ of positive y–smooth integers $\leq x$ we use the approximation

$$\psi(x, y) \approx x \left(\rho(u) + (1 - \gamma) \frac{\rho(u-1)}{\log x} \right), \tag{4.2}$$

that descends from the relation

$$\psi\left(x, x^{1/u}\right) = x \left(\rho(u) + (1 - \gamma) \frac{\rho(u-1)}{\log x} \right) + \mathcal{O}\left(\Delta\left(x, x^{1/u}\right)\right), \tag{4.3}$$

where $x \to \infty$ and Δ is defined by

$$\Delta\left(x, x^{1/u}\right) = \begin{cases} \frac{x^{1/u}}{\log x} + \frac{x}{\log^2 x} & \text{for} \quad 1 < u \leq 2, \\ \frac{x}{\log^{3/2} x} & \text{for} \quad u > 2. \end{cases}$$

For details on the function ψ we refer to the comprehensive bibliography in Norton's memoir [32].

In the sequel we also need an approximation of the number of smooth integers in an interval. Hildebrand and Tenenbaum [18, p. 270] proved that, under the assumption of the Riemann hypothesis, for any fixed ϵ, $0 < \epsilon < 1$, we have

$$\psi\left(x + \frac{x}{z}, y\right) - \psi(x, y) = \frac{\log(1 + y/\log x)}{z \log y} \, \psi(x, y) \left(1 + \mathcal{O}_\epsilon\left(\frac{1}{z} + \frac{\log\log(1+y)}{\log y}\right)\right), \tag{4.4}$$

uniformly in the range

$$x \geq 2, \quad (\log\log x)^{2/3+\epsilon} < \log y \leq (\log x)^{2/5}, \quad 1 \leq z \leq R(x,y)^{-1}. \tag{4.5}$$

Here $R(x,y) = \exp(-y^{1/2-\epsilon}) + \exp(-b_0 u \log^{-2} 2u)\log y$, where b_0 is some positive absolute constant and $u = \log x/\log y$.

We have $\log(1+y/\log x) \approx \log y - \log\log x$ and $\log\log(1+y) \approx \log\log y$. Substituting this and approximation (4.2) into (4.4) it follows that

$$\frac{z}{x}\left\{\psi\left(x+\frac{x}{z},y\right)-\psi(x,y)\right\} \approx$$
$$\left(1-\frac{\log\log x}{\log y}\right)\sigma(x,y,z)\left(1+c_1(\epsilon)\frac{1}{z}+c_2(\epsilon)\frac{\log\log y}{\log y}\right), \tag{4.6}$$

where the function σ is defined by

$$\sigma(x,y,z) = \rho(u) + (1-\gamma)\frac{\rho(u-1)}{\log x},$$

and the $c_i(\epsilon)$ are numbers depending on ϵ.

To test approximation (4.6) (with appropriate values for the numbers $c_i = c_i(\epsilon)$) we sieved y–smooth integers from the interval $[x, x+\Delta]$ for various values of x, y, and Δ. We chose x and y such that their values corresponded to the order of magnitude of polynomial values and values of B respectively, that we used in our experiments described in Section 4.7. In the experiments described in this section $1/z = \Delta/x$ is negligible compared with $\log\log y/\log y$. Using the least squares method we get the number $\tilde{c}_2 = 1.116$ that yields good approximations when using formula (4.6) with $c_2 = \tilde{c}_2$ (and $c_1 = 0$). In Table 4.2 we list the results. (The terms 2×10^5 and 2×10^6 were added to simplify the sieve program.)

x	y	Δ	(4.6)	sieved	quotient
10^{27}	5×10^4	$10^8 + 2 \times 10^5$	3606	3521	1.024
10^{35}	3×10^5	$10^8 + 2 \times 10^5$	527	529	0.996
10^{40}	8×10^5	$10^8 + 2 \times 10^5$	159	149	1.067
10^{45}	6.5×10^5	$10^{11} + 2 \times 10^6$	6771	6818	0.993
10^{50}	8.5×10^5	$10^{11} + 2 \times 10^6$	646	666	0.970
10^{50}	10^6	$10^{11} + 2 \times 10^6$	912	928	0.983

Table 4.2: Tests of approximation (4.6) that estimates the number of y–smooth integers in the interval $[x, x+\Delta]$. The numbers c_1 and c_2 are 0 and 1.116 respectively, and $z = x/\Delta$.

We conclude that approximation (4.6) (and thus approximation (4.2)) is useful in practice in the range of our interest, even if we are not in the range (4.5).

4.4 Complete relations per polynomial

We show how we approximate the number $t_{1,1}$ of complete relations of the right upper branch Γ_1. We divide the interval $[e^T, S]$ in N subintervals $[y_i, y_{i+1}]$ $(i = 0, 1, \ldots, N-1)$ in the following way. Let $h = (\log S - T)/N$, $f_i = T + ih$ and choose $y_i = e^{f_i}$ $(i = 0, 1, \ldots, N-1)$. Hence $y_{i+1} = y_i + \frac{y_i}{z}$, where $z = (e^h - 1)^{-1}$.

To apply (4.6) with an appropriate choice of the numbers c_1 and c_2, we certainly must have $z \geq 1$ so that $e^h - 1 \leq 1$. This means that $N \geq (\log S - T)/\log 2$. Since $S \approx M\sqrt{n/2}$, $T \approx \log(\frac{1}{3}M\sqrt{n/2}) - \log L$, and in practice $L \geq 10B$ and $B \geq 10^5$, N must be larger than $(\log 3 + 6\log 10)/\log 2 \approx 21.5$. Our calculations indicate that $N = 100$ is a safe lower bound. We may choose N much larger so that $1/z$ becomes much smaller (see the end of Section 4.3), but then our algorithm to predict $t_{1,1}$ becomes too slow. Instead we stick to $N = 100$ and take the error constants in approximation (4.6) into account.

Let $x_i \in [-M, M]$ be the number (not necessarily an integer) such that $(x_i, y_i) \in \Gamma_1$. Hence $W(x_i) = y_i = e^{f_i}$. For the slope s_i of the chord from (x_i, y_i) to (x_{i+1}, y_{i+1}) we have $s_i = (y_{i+1} - y_i)/(x_{i+1} - x_i)$.

Let Y be a positive number and let $t_{1,1}^{(Y)}$ $\left(t_{2,1}^{(Y)}\right)$ denote the number of (in)complete relations $y = W(x)$ with $(x, y) \in \Gamma_1$ and $y \leq Y$. Clearly we have

$$t_{1,1} = \sum_{i=0}^{N-1} (t_{1,1}^{(y_{i+1})} - t_{1,1}^{(y_i)}). \tag{4.7}$$

Now we investigate the smoothness probability of polynomial values. Approximation (4.6) estimates the number of *random* smooth integers in an interval and thus we cannot simply apply (4.6) to *special* smooth numbers among polynomial values. Peter L. Montgomery [29] proposed an elegant way to compare the smoothness probabilities of W–values and random numbers. The idea is as follows.

We compute the expected contribution of a prime $p \leq B$ to $W(x)$. Let $p \leq B$ be a prime not dividing the discriminant of W, i.e., p is not a divisor of $4n$. For those primes p we define r_p as the number of roots in the interval $[0, p-1]$ of the congruence equation $W(x) \equiv 0 \pmod{p}$. We have $r_p = 2$ or 0 according as n is a quadratic residue mod p or not. Any root modulo p

corresponds to a unique root mod p^j for any $j > 1$ via Hensel lifting. Hence the expected factor contribution of p to $W(x)$ is

$$p^{r_p\left(\frac{1}{p}+\frac{1}{p^2}+\cdots\right)} = p^{r_p/(p-1)}.$$

We do not sieve with the prime divisors of n and so we put $r_p = 0$ if p divides n. Since $n \equiv 1 \pmod 8$ the expected contribution of prime 2 is

$$2^{2\left(\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\cdots\right)} = 2^2.$$

Thus, if we finally define $r_2 = 2$ then the estimated logarithmic norm after sieving by elements in the factor base is

$$\log W(x) - \sum_{p \le B} r_p \frac{\log p}{p-1}.$$

Since the corresponding value for a random number y equals $\log y - \sum_{p \le B} (\log p)/(p-1)$, we assume that the numbers $W(x)$ are about as smooth as random integers with logarithmic norm $\alpha + \log W(x)$, where

$$\alpha = \sum_{p \le B} (1 - r_p) \frac{\log p}{p-1}. \tag{4.8}$$

The probability that for a random integer y, with $y_i \le y \le y_{i+1}$, there exists an integer x such that (x, y) is a member of Γ_1 is approximately $1/s_i$. Hence, using the correction term α in (4.6), it follows that

$$\begin{aligned} \frac{t_{1,1}^{(y_{i+1})} - t_{1,1}^{(y_i)}}{y_{i+1} - y_i} &\approx \frac{1}{s_i}\left(1 - \frac{\log g_i}{\log B}\right)\left(\rho(v_i) + (1-\gamma)\frac{\rho(v_i - 1)}{g_i}\right) \\ &\quad \times \left(1 + c_1 \frac{1}{z} + c_2 \frac{\log\log B}{\log B}\right), \end{aligned} \tag{4.9}$$

where $g_i = \alpha + f_i$ and $v_i = g_i / \log B$. Note that $y_i/(s_i z) = x_{i+1} - x_i$. Combining this approximation with formula (4.7), we obtain an approximation of $t_{1,1}$.

To approximate $t_{1,2}$, we replace Γ_1 by Γ_2, S by R, y_i by $|\, y_i \,|$, $x_{i+1} - x_i$ by $x_i - x_{i+1}$ and proceed as above. Since $W(x)$ is almost symmetric around the y–axis by construction, we have $t_{1,3} \approx t_{1,1}$ and $t_{1,4} \approx t_{1,2}$.

4.5 Incomplete relations per polynomial

We show how to approximate the number $t_{2,1}$ of incomplete relations from branch Γ_1. We have

$$t_{2,1} = \sum_{i=0}^{N-1} (t_{2,1}^{(y_{i+1})} - t_{2,1}^{(y_i)}). \tag{4.10}$$

Since the probability that a W–value is divisible by a prime q is r_q/q, we have

$$t_{2,1}^{(Y)} \approx \sum_{B<q\leq L} \frac{r_q}{q}\, t_{1,1}^{(Y/q)},$$

where the summation ranges over all primes q between B and L. Write $g_{i,q} = \alpha + f_i - \log q$ and $v_{i,q} = g_{i,q}/\log B$. We then obtain

$$\begin{aligned} t_{2,1}^{(y_{i+1})} - t_{2,1}^{(y_i)} &\approx \sum_{B<q\leq L} \frac{r_q}{q} \left(t_{1,1}^{(y_{i+1}/q)} - t_{1,1}^{(y_i/q)}\right) \\ &\approx (x_{i+1} - x_i) \sum_{B<q\leq L} \frac{r_q}{q} \left(1 - \frac{\log g_{i,q}}{\log B}\right) \left(\rho(v_{i,q}) + (1-\gamma)\frac{\rho(v_{i,q}-1)}{g_{i,q}}\right) \\ &\quad \times \left(1 + c_1 \frac{1}{z} + c_2 \frac{\log\log B}{\log B}\right), \end{aligned} \tag{4.11}$$

where we used (4.9) with f_i replaced by $f_i - \log q$ and applied the definition of s_i. Approximation (4.11) together with equation (4.10) yields an approximation of $t_{2,1}$. To compute approximations $t_{2,i}$ $(i = 2, 3, 4)$ we make similar adjustments as at the end of Section 4.4.

4.6 The total number of complete relations

We wish to compute E, the expected number of complete relations coming from a given number of r partial relations. Let $\mathcal{Q} = \{\text{primes } q \mid B < q \leq L,\ (\frac{n}{q}) = 1\}$. The elements of $\mathcal{Q}$ are called *large primes*. Let P_q be the probability that a large prime q occurs in a partial relation. Lenstra and Manasse [25] assume that

$$P_q \approx q^{-\beta} \Big/ \sum_{p\in\mathcal{Q}} p^{-\beta} \tag{4.12}$$

for some positive constant $\beta < 1$ that should be determined experimentally. They report that $\beta \in [\frac{2}{3}, \frac{3}{4}]$ gives a reasonable fit with their experimental results. Denny [15, pp. 44–49] takes $\beta = 0.775$.

From [25] it follows that

$$E = r - |\mathcal{Q}| + \sum_{q \in \mathcal{Q}} (1 - P_q)^r.$$

We apply the binomial formula of Newton and use approximation (4.12) to find:

$$E \approx \sum_{i=2}^{r} (-1)^i \binom{r}{i} \left(\sum_{q \in \mathcal{Q}} q^{-\beta} \right)^{-i} \sum_{q \in \mathcal{Q}} q^{-\beta i}. \tag{4.13}$$

Since $\pi(t) \sim t/\log t$ as $t \to \infty$, we have

$$\sum_{x' \leq p \leq x} p^{-u} \approx \int_{x'}^{x} t^{-u} d(t/\log t)$$

(p prime, $x \in \mathbf{R}_{\geq 2}$, $u \in \mathbf{R}_{>0}$) provided that x/x' is not too small. Hence for $u > 0$ we have

$$\sum_{q \in \mathcal{Q}} q^{-u} \approx \frac{1}{2} \int_{B}^{L} t^{-u} d(t/\log t). \tag{4.14}$$

To compute the last integral we first use partial integration and then use the substitution $s = (1-u)\log t$. We get

$$\begin{aligned} \int_{B}^{L} t^{-u} d(t/\log t) &= L^{1-u}/\log L - B^{1-u}/\log B \\ &\quad + u\{\mathrm{Ei}((1-u)\log L) - \mathrm{Ei}((1-u)\log B)\}, \end{aligned} \tag{4.15}$$

where Ei is the exponential integral defined by $\mathrm{Ei}(x) = \int_{-\infty}^{x} (e^s/s) ds$. Now combine (4.13), (4.14), and (4.15) for the appropriate choices of u to get an approximation for E. In approximation (4.13) we sum from $i = 2$ to $i = 5$ and forget about the higher order terms to get a formula for an approximation of E that we can use in practice (given B, L, r, and β).

The experiments summarized in Table 4.3 show that our approximation works well if we choose $\beta = 0.73$. The table shows, for each example run, the number r of partial relations, the estimated number of complete relations derived from these partial relations, and the actual number of complete relations. An approximation of E may be used to *predict* the computing time. We determined β as follows. We wrote a program in Maple that, given β, computes the absolute value of the difference of the actual number of complete relations and the estimated number of complete relations for each of fifteen test numbers we took. Then we summed the fifteen absolute values of the differences, thus obtaining for each β a sum of absolute values. The smallest sum was attained at $\beta = 0.73$.

n	$\frac{B}{10^5}$	L/B	r	$\tilde{n}_{c,1}$	$n_{c,1}$
C75	3	20	37472	4966	4790
C80	1	60	15918	1209	1121
C80	3	167	68195	4150	4113
C84	8	25	96138	11148	10894
C88	5	100	94651	6736	6605
C88	7.5	100	148403	11211	11455
C88	7.5	100	158214	12657	12830
C88	7.5	100	146983	11008	11051
C88	7.5	100	150327	11488	11498
C88	7	100	148016	11827	12116

Table 4.3: Results of experiments to determine β. For ten composite numbers and bounds B, L, we list the number r of partial relations and the estimated ($\tilde{n}_{c,1}$) and actual ($n_{c,1}$) number of complete relations derived from the partial ones. As usual, Cx denotes a composite number with x decimal digits.

4.7 Numerical results

We list the results of our experiments in Tables 4.4, 4.5, and 4.6. For each number to be factored we sieved thousands of polynomials and computed the average number of (in)complete relations obtained per polynomial. The number of polynomials we sieved is denoted by r, the estimated values of t_1 and t_2 are written as $\tilde{t}_1$ and $\tilde{t}_2$ respectively. By r.e. in the tables we mean relative error. We picked *one* polynomial to compute $\tilde{t}_1$ and $\tilde{t}_2$ as described above, since different polynomials in one experiment turned out to give almost the same values for $\tilde{t}_1$ and $\tilde{t}_2$.

We selected polynomials such that the leading coefficient a of each polynomial was the square of an integer co–prime with primes in the factor base.

We computed the Taylor series of ρ up to degree 55. Since we had to compute many values of the Dickman–De Bruijn function we precomputed a table of values $\rho(u)$ from $u = 2$ to $u = 10$ using a step size of $1/2^{11}$. Using linear interpolation we then approximated $\rho(u)$ for a particular value of u. The number N of subintervals of one branch was chosen te be 100.

To determine numbers c_1 and c_2 in approximations (4.9) and (4.11) we chose sample numbers to be factored (listed in the Appendix), determined the actual number of complete and incomplete relations per polynomial and used the least squares method to minimize the sum of the squares of the

relative errors. We found $c_1 = -0.4813$ and $c_2 = 1.344$ for approximation (4.9) and $c_1 = 1.688$ and $c_2 = -2.372$ for approximation (4.11).

The program for the approximation of complete relations was written in Maple V Release 3; for the incomplete relations we wrote a program in Fortran.

4.7.1 Results for the complete relations

We did experiments for ten values of $(n/m, \sqrt{a}, b)$. (Note that coefficient c is determined by a and b since the discriminant of each polynomial is equal to $4n$.) These values are listed in the Appendix. Tables 4.4 and 4.5 contain the chosen values of the parameters, the resulting value of α and the actually found value of t_1 compared with the calculated estimate $\tilde{t}_1$.

Number:	1a	2a	3a	4a	5a
m	1	41	1	47	71
$B/10^5$	2	4	3	4	3
$M/10^6$	1	0.5	1	1	0.5
T	76.55	80.32	82.42	84.74	87.46
r	3400	13000	25400	16400	117100
α	−2.605	−2.150	−0.5899	−2.373	−1.881
$\tilde{t}_1$	2.148	1.241	0.5849	0.8715	0.1099
t_1	2.604	1.206	0.5089	0.9684	0.1004
r.e. (%)	−17.5	2.89	14.9	−10.0	9.41

Table 4.4: Estimated ($\tilde{t}_1$) and actual (t_1) number of complete relations per polynomial. The sample numbers are listed in the Appendix.

4.7.2 Results for the incomplete relations

We did experiments for six tuples $(n/m, \sqrt{a}, b)$. Again, see the Appendix for the actual values. Table 4.6 contains the parameter values and the results of the experiments.

4.8 Determining good parameters

In practice the sieve phase dominates the run time of the algorithm; it takes more than 90 % of the total time. Therefore we only consider the

Number:	6a	7a	8a	9a	10a
m	41	79	13	1	29
$B/10^5$	7	5	5	8	9.5
$M/10^6$	2	2.5	2.5	3	5
T	99.49	84.87	86.07	104.6	113.6
r	22314	292280	287312	65260	36785
α	−2.652	−1.972	−1.293	−2.145	−1.592
$\tilde{t}_1$	0.1332	0.02646	0.02702	0.06321	0.01360
t_1	0.1423	0.03058	0.03009	0.06281	0.01279
r.e. (%)	−6.41	−13.5	−10.2	0.638	6.31

Table 4.5: Estimated ($\tilde{t}_1$) and actual (t_1) number of complete relations per polynomial. The sample numbers are listed in the Appendix.

amount of work done in the sieve part of the algorithm. The determination of good parameter values depends heavily on the computer used and the implementation of the algorithm.

The amount of work is approximately proportional to the number of sieve updates (additions of logarithms to the elements of the sieve array). Per root and per factor base element q we have to apply the sieve updates on the $2M + 1$ cells of the sieve interval, using stride q. This means that the number of sieve updates per polynomial is approximately equal to

$$4M \sum_{q \in \mathcal{F}} \frac{1}{q}.$$

The total number of complete relations after processing r polynomials is approximately $rt_1 + E(rt_2)$. Since we have to generate at least $1 + |\mathcal{F}|$ complete relations, an approximation of the minimal number of polynomials needed is the solution r_0 of the equation

$$r_0 t_1 + E(r_0 t_2) = 1 + |\mathcal{F}|. \tag{4.16}$$

We determine r_0 by using binary search in some interval. The total amount of work is approximately

$$4Mr_0 \sum_{q \in \mathcal{F}} \frac{1}{q} \tag{4.17}$$

sieve updates and the expression is dependent on B, L, and M. By varying the parameters in some interval, one can compute the total amount of work and thus determine good parameters.

Number:	1b	2b	3b	4b	5b	6b
m	47	79	13	1	23	1
$B/10^5$	4	5	5	8	9.5	10
$L/10^6$	6	10	10	16	14.25	10
$M/10^6$	1	2.5	2.5	2	10	10
T	88.02	84.87	86.07	97.64	107.1	113.8
r	34911	292280	287312	36400	17842	23087
α	−1.865	−1.972	−1.293	−2.145	−2.072	−0.9389
$\tilde{t}_2$	0.3908	0.2366	0.2296	0.2796	0.1807	0.03370
t_2	0.5052	0.2571	0.2569	0.3063	0.1876	0.03180
r.e. (%)	−22.6	−7.97	−10.6	−8.71	−3.66	6.08

Table 4.6: Estimated ($\tilde{t}_2$) and actual (t_2) number of incomplete relations per polynomial. The sample numbers are listed in the Appendix.

On a computer without hierarchical memory (cache or virtual memory), for example the Cray C90 supercomputer, the CPU–time is a linear function in (4.17). Of course the same holds for implementations on computers with virtual memory, as long as page–faults do not occur during the sieving process. ("Pages" are used to quickly determine the addresses in memory of data to be dragged to cache.)

Most people do not have access to a supercomputer and use an implementation on workstations (or PCs) with cache. In this case the CPU–time is dominated by cache effects. It is difficult to give one formula that gives the CPU–time in terms of the parameters since in general different computers have different cache strategies.

We give an example that illustrates how to determine good parameters. For a given choice of the parameters we estimate the number of complete and incomplete relations generated by one polynomial by formulae (4.9) and (4.11) and we estimate the total number of polynomials needed by solving equation (4.16). Next we do the actual sieve run for *one* polynomial and measure the CPU–time. Thus we have an estimation of the total sieve time. To verify our estimates we also carry out the complete sieve run for each choice of the parameters. In practice of course, this is only done for the final choice of parameters derived from our estimates. The sample number is

C62 10 5783259093 2620060454 1346963019 3620363971\
1810100364 0923795313 (4.18)

with $m = 1$, $M = 2.5 \times 10^5$, $T = 68.12$, $\alpha = -1.522$, $\sqrt{a} = 429\ 1273798937$, and $b = 27443\ 6107325508\ 5474186145$.
We vary the most important parameter B in the range $[10^5, 5 \times 10^5]$ with step size 5×10^4 and take $L = 10 \cdot B$ for each choice of B. For simplicity the other parameters are kept fixed for each choice of B. Table 4.7 contains columns for the estimated and actual number r_0 of polynomials needed and the total sieve time in seconds. The experiments were carried out on a Silicon Graphics Indy workstation with one 100 MHz R4000 processor and 8 Kb data cache size. (Since we used an implementation written for the Cray Y–MP vector computer, the code was not optimal for usage on workstations so a better performance should be possible.)

$B/10^5$	$\tilde{r}_0$	r_0	$\tilde{T}_s$	T_s
1	2377	2055	16654 s	14398 s
1.5	1641	1425	11974 s	10395 s
2	1317	1155	10119 s	8874 s
2.5	1127	990	9140 s	8033 s
3	1003	885	8671 s	7655 s
3.5	918	825	8529 s	7674 s
4	855	765	8589 s	7689 s
4.5	804	720	8730 s	7825 s
5	764	690	9000 s	8135 s

Table 4.7: Estimated ($\tilde{r}_0$) and actual (r_0) minimal number of polynomials needed and the estimated ($\tilde{T}_s$) and actual (T_s) sieve times. Test runs with (4.18) for various values of B. For each choice of B we take $L = 10 \cdot B$. Other relevant quantities involved are: $m = 1$, $M = 2.5 \times 10^5$, $T = 68.12$, $\alpha = -1.522$. These quantities are kept fixed for each choice of B.

The relative error in the estimation of the total sieve time is less than 16 %. We have plotted the estimated and actual sieve time. The estimated times are systematically higher than the actual times, but since the *shape* of the two graphs is the same, the minimum of the estimated time graph lies close to that of the actual time graph. The graphs show that $B = 3 \times 10^5$ is a good choice, but some larger values are also acceptable since from there the sieve time does not fluctuate too much. If B becomes smaller than 2×10^5 then the sieve time increases considerably. Therefore, to be sure, one might choose a B that is somewhat larger than the estimated optimal choice.

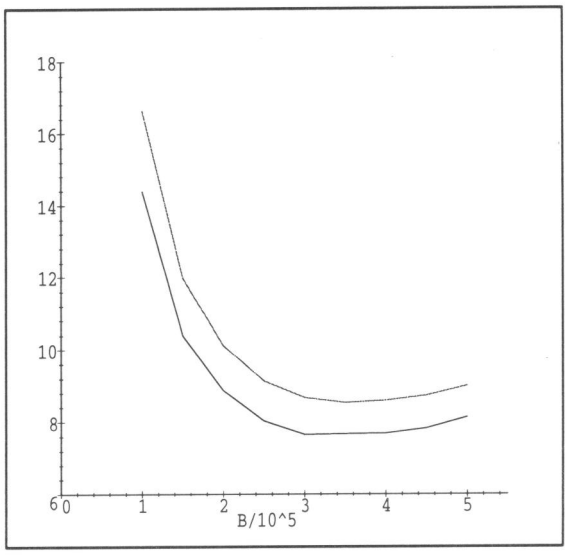

Figure 4.1: Estimated (upper graph) and actual (lower graph) sieve times (/1000 s) as functions of B; see Table 4.7.

4.9 Conclusions

We have given expressions to approximate the average number of complete and incomplete relations per polynomial we find using the one large prime variant of MPQS. Next, the number of complete relations descendent from a given number of incomplete relations is estimated by using a known formula. From these results we can derive a good approximation of the total number of polynomials needed and hence we are able to estimate the total sieve time after processing one polynomial. The prediction formulae may be used to determine good parameters.

For numbers with 65–100 decimal digits our experiments indicate that the average relative error of the estimations of the number of (in)complete relations per polynomial is about 10 %.

An example suggests that we may estimate the total sieve time with a relative error of less than 20 % only on the basis of a test run on *one* polynomial. The example, used to find a good value for B, yields a range of B-values where the actual sieve time is close to minimal.

Along the way we tested approximation formula (4.6) for the *total* number of smooth integers in an interval and observed that the approximation worked well in the range of our interest, even if conditions (4.5) were not satisfied completely. In the experiments with more than 500 sieved numbers the error of the estimate was less than 5 %. From this it follows that the classical formula (4.3) for the approximation of the total number of smooth numbers below some bound is also useful in practice.

Appendix A

In 1925 Lt.–Col. Alan J. C. Cunningham and H. J. Woodall started collecting the factors of numbers of the form $a^n \pm 1$, for various values of a [13]. Such numbers often occur in mathematics. The multiplicative group of the finite field with a^n elements (where a is prime) has order $a^n - 1$. Also, if a is prime, the sum of divisors of a^n is $\sigma(a^n) = (a^{n+1} - 1)/(a - 1)$. The computation of tables of factors of these numbers is referred to as the *Cunningham project*, that is currently under the auspices of Richard Brent, Peter Montgomery, and Herman te Riele [6]. For the history, see the introduction in [8].

We factored many numbers (with MPQS2) to update the Cunningham Table. We also factored some numbers of the form $a^n \pm 1$ that are *outside* the range covered by [6]. Hans Riesel, e.g., asked us to do # 45 in Table A.2.

All the numbers first endured factoring algorithms such as the Pollard $p \pm 1$ methods and the elliptic curve method to divide out possible small factors. Only when these methods failed to factor a number or composite cofactor, it was passed to the quadratic sieve.

Table A.1 shows comprehensive statistics for 8 numbers in the range of 66–83 digits done with a single workstation. Table A.2 contains information about 73 numbers in the 67–88 digit range done with the Cray C90 vector computer. For these numbers we used the unblocked implementation of MPQS2. We factored about 125 other numbers with more than 84 digits with the self–initializing blocked variant of MPQS2. The statistics of 64 of these numbers are listed in Table A.3. The sieving was done in parallel on 69 workstations at University of Leiden. The recovery files were stored locally. Since we sieved during the night and in the weekends, we did not or could not gather them to check whether enough relations had been generated yet. Hence, often we sieved too long.

#	n	prime factor(s)
1	C66: $77^{53}+1$ = P31 · P35	P31 = 8508101816450689975658227843439
2	C67: $58^{88}+1$ = P26 · P41	P26 = 62057338333442627487392257
3	C67: $62^{89}-1$ = P31 · P37	P31 = 3916898265747514256035560079891
4	C75: $70^{87}+1$ = P29 · P46	P29 = 56476537654063551106920429541
5	C79: $72^{118}+1$ = P38 · P42	P38 = 160594909070093212254803474806878324 41
6	C82: $84^{71}+1$ = P33 · P50	P33 = 133184106044570646620234096956423
7	C82: $80^{99}+1$ = P32 · P51	P32 = 11935171798229644025656192643827
8	C83: $92^{87}+1$ = P23 · P61	P23 = 10127992394070979564027

Table A.1: Parameter choices, timings, and factors for numbers ranging from 66 to 83 decimal digits, factored with MPQS2 (unblocked) on a 100 MHz SGI workstation. Key: n = number to be factored. Cx: y means a composite factor of y having x decimal digits; Px means a prime factor having x decimal digits; $d = {}^{10}\log n$; B = upper bound for primes in the factor base; L^2 = upper bound for the input R to SQUFOF (yielding a pp–relation);

#	n	prime factor(s)
1	C67: $89^{64}+1$ = P24 · P44	P24 = 153316525308739316934017
2	C69: $50^{122}+1$ = P30 · P40	P30 = 276832194921994230575098974137
3	C75: $101^{41}+1$ = P32 · P43	P32 = 21587227703328821952030527314507
4	C75: $110^{41}+1$ = P16 · P25 · P35	P16 = 3850561614882023, *)
5	C75: $110^{47}+1$ = P24 · P51	P24 = 728424414211828929294823
6	C75: $35^{147}+1$ = P35 · P40	P35 = 86052439411099140168070862933143801
7	C75: $53^{59}-1$ = P24 · P51	P24 = 943970114867362247759443
8	C78: $19^{165}+1$ = P28 · P50	P28 = 2481953419044452308291386601
9	C78: $51^{102}+1$ = P30 · P48	P30 = 459028910227193494771112394289
10	C80: $86^{58}+1$ = P33 · P47	P33 = 129094951090723152084884804969621
11	C80: $75^{64}+1$ = P32 · P48	P32 = 68799038786512319388821350925569
12	C80: $59^{85}-1$ = P36 · P44	P36 = 192052183634195717382812875959337681

Table A.2: Parameter choices, timings, and factors for numbers ranging from 67 to 88 decimal digits, factored with MPQS2 (unblocked) on a Cray C90

#	d	$\frac{B}{10^5}$	n_f	$\frac{L}{B}$	$\frac{M}{10^5}$	n_c	n_1	$n_{c,1}$	n_2	$n_{c,2}$	T_s
1	65.56	0.8	3911	11.25	2	1493	9753	1715	4102	710	5.8 h
2	66.17	0.8	3908	10	1.5	1452	9433	1766	3697	693	4.8 h
3	66.83	0.8	3984	10	2	1214	9952	2139	4238	637	14.2 h
4	74.15	3	13045	20	6	4840	37472	4790	26391	3424	55.4 h
5	78.76	3	12898	30	5	4444	44583	5104	29653	3355	123.0 h
6	81.54	5	20812	20	5	7992	63176	8471	33614	4351	173.0 h
7	81.70	4.5	18961	20	4.5	6796	55435	7229	38950	4942	198.0 h
8	82.89	5	20861	20	8	7387	62346	8229	40035	5250	273.0 h

n_f = number of primes in the factor base; $[-M, M]$ = sieve interval; n_c = number of complete relations found immediately; n_1 = number of partial relations; $n_{c,1}$ = number of complete relations coming from partial relations; n_2 = number of pp–relations; $n_{c,2}$ = number of complete relations coming from pp-relations; T_s = sieve CPU time. The small primes variation parameter QT is always 40.

#	d	$\frac{B}{10^5}$	n_f	$\frac{L}{B}$	$\frac{M}{10^5}$	n_c	n_1	$n_{c,1}$	n_2	$n_{c,2}$	T_s
1	66.80	2	8881	30	25	2945	27673	2762	31855	3347	0.36 h
2	68.74	2.5	11086	20	5	3988	30107	3631	27746	3476	0.46 h
3	74.20	3.16	13623	20	6.31	4921	38371	4889	29855	3822	1.22 h
4	74.51	3.16	13625	20	6.31	5503	42284	5844	17604	2297	1.16 h
5	74.69	1	4790	60	5	1005	17630	1320	29502	2465	2.42 h
6	74.83	3	12892	17	25	4697	37137	5388	19447	2820	1.20 h
7	74.92	2.5	11086	36	25	3339	35899	3335	43531	4382	1.91 h
8	77.37	5	20972	20	25	7152	54706	6444	60361	7393	1.84 h
9	77.56	5	20888	30	30	7518	65930	6980	60042	6453	1.41 h
10	79.04	5	20597	30	30	6596	61563	6201	76295	7828	2.43 h
11	79.17	4	16927	20	1	5619	45717	5584	48399	6279	3.29 h
12	79.17	5	20895	20	3	6457	72272	11650	37114	2802	2.68 h

vector computer. Key as in table A.1. Continued overleaf.
*) P25 = 77975982398530740576552l9.

#	n	prime factor(s)
13	C80: $76^{123}+1$ = P28 · P53	P28 = 160247580154635097509486030 7
14	C80: $84^{87}-1$ = P40 · P41	P40 = 2904043752413366850400636076474517615769
15	C81: $18^{103}-1$ = P35 · P47	P35 = 15936754604932361311519937275763087
16	C83: $82^{68}+1$ = P40 · P43	P40 = 9241855378580566956862595601843404638609
17	C83: $93^{71}+1$ = P34 · P50	P34 = 18715988916952079528029392484745 57
18	C84: $89^{67}-1$ = P41 · P44	P41 = 17345460386856072657168883886351357651503
19	C84: $74^{91}-1$ = P31 · P54	P31 = 630045464973369109978612017864 7
20	C85: $69^{117}+1$ = P42 · P43	P42 = 553775456930001686459646662784000439421893
21	C85: $98^{91}+1$ = P39 · P47	P39 = 150856027763097994901861400756223948651
22	C85: $80^{58}+1$ = P42 · P44	P42 = 587407531780545617292693056474932755332969
23	C85: $56^{64}+1$ = P43 · P43	P43 = 1120971223480359091305712645673434758493441
24	C85: $39^{111}-1$ = P32 · P54	P32 = 3866190103786178771734741205 0407
25	C85: $77^{95}-1$ = P34 · P52	P34 = 1254200040785197567017611121581711
26	C86: $18^{111}+1$ = P35 · P51	P35 = 57095169829153516132919139336069139
27	C86: $76^{59}+1$ = P39 · P47	P39 = 471586815074704431240140019672222092489
28	C86: $20^{97}+1$ = P34 · P52	P34 = 26453329120142876693394950899515 67
29	C86: $93^{99}-1$ = P31 · P55	P31 = 346673259388800825479161336008 1
30	C86: $58^{93}-1$ = P32 · P54	P32 = 75701865042739143157590250368211
31	C86: $56^{96}+1$ = P39 · P47	P39 = 232559086557407467762901333407938321409
32	C86: $92^{84}+1$ = P43 · P43	P43 = 24651527156587484288308809948243436390198 33
33	C86: $67^{99}-1$ = P34 · P52	P34 = 2515208214206285121254951932641469
34	C86: $13^{138}+1$ = P29 · P57	P29 = 54836637716450236990971812089
35	C86: $59^{89}-1$ = P31 · P55	P31 = 26899414244883480238486498083 89
36	C86: $21^{123}+1$ = P39 · P47	P39 = 380770063539669474313312691529545132713
37	C86: $38^{81}-1$ = P36 · P50	P36 = 51166207516397076206041753843648 4323
38	C86: $31^{117}-1$ = P39 · P47	P39 = 250630033376957433234617073114910871767
39	C86: $50^{96}+1$ = P35 · P51	P35 = 367741123007653820679611686528008 97
40	C86: $96^{95}+1$ = P28 · P58	P28 = 241847699068879601458189083 1
41	C86: $24^{130}+1$ = P36 · P50	P36 = 6849899286441940017850759226564468 41
42	C86: $93^{53}+1$ = P38 · P49	P38 = 191926998695502533890959785501678 28173
43	C86: $98^{59}+1$ = P32 · P55	P32 = 290370474482098105894756472922 91
44	C86: $80^{65}+1$ = P31 · P55	P31 = 341687167491915869952874280124 1

#	d	$\frac{B}{10^5}$	n_f	$\frac{L}{B}$	$\frac{M}{10^5}$	n_c	n_1	$n_{c,1}$	n_2	$n_{c,2}$	T_s
13	79.39	3	13001	166.7	3	3739	68195	4113	72708	5157	2.27 h
14	79.87	3	13011	166.7	3	3323	64308	3624	91150	6084	3.41 h
15	80.86	5	20819	20	6	6925	57619	7050	55281	6877	3.36 h
16	82.82	6	24598	20	2.5	8522	68723	8378	59901	7713	4.82 h
17	82.91	7	28413	20	2.5	11451	87010	11694	40636	5271	4.38 h
18	83.66	8	32104	25	2.5	11419	96138	10894	85260	9807	5.46 h
19	83.98	7	27980	25.7	2.5	10594	93766	11327	51233	6070	6.59 h
20	84.10	5	20713	20	2.5	6175	51592	5808	76377	8732	5.6 h
21	84.35	5	20741	20	2.5	6865	60444	7638	72201	6256	9.8 h
22	84.80	5	20744	20	2.5	7809	57576	7457	44022	5481	9.8 h
23	84.87	5	20790	20	2.5	7153	61546	7923	43044	5721	8.4 h
24	84.92	5	20930	20	2.5	6434	52315	5865	75217	8614	6.0 h
25	84.99	5	20749	20	2.5	7106	58607	7259	53507	6389	6.8 h
26	85.02	5	20675	20	2.5	6982	61080	7920	64746	5774	9.8 h
27	85.02	5	20792	20	2.5	6679	58782	7268	81258	6853	11.0 h
28	85.05	5	20887	20	2.5	7754	65228	8990	46265	4178	8.4 h
29	85.11	5	20810	20	2.5	4923	43182	4064	280566	11857	8.4 h
30	85.11	5	20841	20	2.5	5615	50651	5434	182705	9822	10.7 h
31	85.12	6	24641	20	2.5	8518	67320	9253	73153	6953	10.0 h
32	85.12	5	20651	20	1.5	7269	64239	8799	48843	4625	9.5 h
33	85.14	5	20812	20	1.5	5064	43981	4223	263194	11614	10.7 h
34	85.21	5	20709	20	1.5	6722	56788	6924	92891	7136	8.27 h
35	85.26	5	20859	20	1.5	5101	44412	4378	256996	11412	11.0 h
36	85.26	5	20768	20	1.5	7186	63721	8449	57739	5154	12.4 h
37	85.31	5	20812	20	1.5	5297	45852	4584	185169	10967	7.45 h
38	85.31	5	20576	20	1.5	6107	55044	6362	130553	8115	13.1 h
39	85.33	5	20709	20	1.5	6859	60552	7686	68840	6177	11.5 h
40	85.35	5	20923	20	1.5	6480	55476	6546	127090	7903	10.7 h
41	85.37	5	20672	20	1.5	7221	62980	8435	54345	5029	9.65 h
42	85.42	5	20672	20	1.5	5695	50790	5707	139604	9308	10.4 h
43	85.49	5	20772	20	1.5	7530	63927	8600	51034	4656	11.9 h
44	85.52	5	20634	20	1.5	5556	50383	5456	185347	9653	13.7 h

#	n	prime factor(s)
45	C86: $8^{2^7}+7^{2^7}$ = P42 · P44	P42 = 519975935060346660783986052760977025136897
46	C86: $23^{83}-1$ = P38 · P49	P38 = 27736074503263071062950778805992164759
47	C86: $76^{56}+1$ = P40 · P47	P40 = 4868699568817220592890920460964327586529
48	C86: $47^{67}-1$ = P32 · P55	P32 = 21270964162538089013014983761851
49	C86: $67^{76}+1$ = P42 · P45	P42 = 315618216027848486834301078445774290254513
50	C86: $39^{81}+1$ = P37 · P50	P37 = 24430036165666630699892784411335180 59
51	C86: $22^{95}-1$ = P34 · P52	P34 = 96243579190684035550915123674142 61
52	C86: $76^{117}-1$ = P42 · P45	P42 = 606202897105850025527074421945484005533987
53	C86: $95^{80}+1$ = P38 · P49	P38 = 450897580997918678316374862447596670 41
54	C87: $62^{65}+1$ = P34 · P53	P34 = 14391069229025228424841101554443 91
55	C87: $72^{99}+1$ = P28 · P59	P28 = 80975407891689909106865888 41
56	C87: $92^{85}-1$ = P32 · P56	P32 = 14285278844357974752432939513571
57	C87: $30^{95}+1$ = P35 · P52	P35 = 80451911996934444483653727156040931
58	C87: $50^{100}+1$ = P41 · P46	P41 = 589514788785130719305008867620773920776 01
59	C87: $66^{96}+1$ = P42 · P46	P42 = 153055732248039041786999207837459270270017
60	C87: $19^{101}-1$ = P25 · P62	P25 = 5245647644316863182854571
61	C87: $33^{85}+1$ = P33 · P54	P33 = 24953692198916926106503511225790 1
62	C87: $63^{65}+1$ = P42 · P46	P42 = 10841088997442568505957564739184105515545 1
63	C87: $42^{99}-1$ = P33 · P55	P33 = 23437309093413719343442610084173 9
64	C87: $77^{67}-1$ = P41 · P46	P41 = 75024943244844149373705126243013155715853
65	C87: $84^{59}-1$ = P35 · P53	P35 = 11779548019122302808328920808327631
66	C87: $26^{129}+1$ = P31 · P57	P31 = 3076814278757622588317626405309
67	C87: $33^{111}-1$ = P38 · P50	P38 = 21457939605898871224437297672972660829
68	C87: $86^{84}+1$ = P40 · P48	P40 = 1039512269081394539159468072656199331337
69	C87: $85^{65}+1$ = P40 · P48	P40 = 4645176624103101144238593467706089788481
70	C87: $45^{85}+1$ = P36 · P52	P36 = 2181360904850689209750606257400202 21
71	C87: $87^{93}+1$ = P35 · P53	P35 = 65234702723152738657728499902597613
72	C87: $45^{71}+1$ = P27 · P61	P27 = 69229816187403473081388160 3
73	C88: $19^{168}+1$ = P42 · P47	P42 = 261688712348581672325146786097393313497473

#	d	$\frac{B}{10^5}$	n_f	$\frac{L}{B}$	$\frac{M}{10^5}$	n_c	n_1	$n_{c,1}$	n_2	$n_{c,2}$	T_s	
45	85.53	5	20711	20	2.5	5054	43759	4078	270308	11587	11.4	h
46	85.59	5	20797	20	1.5	7244	63668	8524	61191	5044	12.6	h
47	85.70	5	20712	20	1.5	6637	56910	6862	96694	7226	10.3	h
48	85.72	5	20911	20	1.5	6311	56349	6710	120384	7895	15.4	h
49	85.73	6	26392	2	3	16159	24514	9487	3153	749	13.8	h
50	85.92	6	26363	2	3	16376	24473	9358	7417	631	12.1	h
51	85.93	3	13041	20	1.5	4117	39602	5212	39395	3713	17.4	h
52	85.95	3	13011	20	2.5	4255	40517	5390	24478	3366	20.6	h
53	85.98	5	20756	2.4	2.5	10516	22044	7450	6610	2795	15.7	h
54	86.04	5	20840	20	2.5	7153	62231	8139	63273	5557	14.0	h
55	86.13	5	20838	20	2.5	7009	63089	8220	57513	5620	11.9	h
56	86.16	5	20787	22	2.5	7367	63987	8559	54708	4900	10.8	h
57	86.18	5	20688	20	2.5	7447	64778	8836	47154	4419	10.7	h
58	86.22	5	20852	20	2.5	6202	54180	6282	144069	8376	11.6	h
59	86.22	5	20947	20	2.5	7522	63620	8412	52191	5091	9.35	h
60	86.27	5	20978	40	2.5	6773	79489	8184	75416	6035	14.2	h
61	86.29	5	20797	40	2.5	6387	72868	6909	116000	7520	11.1	h
62	86.38	5	20754	40	2.5	6881	76861	7638	86915	6253	6.72	h
63	86.43	5	20920	40	2.5	7177	76854	7706	82085	6054	8.81	h
64	86.45	5	20631	40	2.5	6329	74485	7262	92362	7046	14.6	h
65	86.63	5	20902	80	2.5	5806	85167	6249	148784	8876	13.8	h
66	86.64	6	24404	100	3	7564	124510	9691	89594	7355	13.2	h
67	86.69	6	24573	100	3	7803	122935	9614	89375	7369	14.1	h
68	86.70	6	24495	100	3	6571	105037	7010	149698	11272	12.1	h
69	86.73	6	24538	100	3	7635	120888	9389	99930	7811	11.5	h
70	86.75	6	24374	100	3	7827	126178	9899	80444	6862	14.7	h
71	86.82	6	24615	100	3	6532	121187	9864	167590	8507	11.8	h
72	86.96	6	24658	100	3	7762	116023	8546	126334	8798	7.66	h
73	87.54	5	20604	100	1	6101	94651	6605	108893	8048	12.1	h

#	n	d	m	$\frac{B}{10^5}$	$\frac{L}{10^8}$	$\frac{p1}{10^8}$	$\frac{p2}{10^8}$	$\frac{M}{10^5}$	QT	RT
1	C85: $96^{83}+1$	84.02	7	6.0	10.0	0.1	1.0	2.5	100	69
2	C85: $45^{130}+1$	84.34	1	8.5	1.0	1.0	1.0	5.0	200	73
3	C85: $38^{125}-1$	84.47	1	6.0	10.0	0.1	1.0	2.5	100	69
4	C85: $33^{92}+1$	84.52	1	6.0	10.0	0.1	1.0	2.5	100	70
5	C85: $15^{103}+1$	84.58	1	5.0	0.1	0.1	1.0	10.0	200	63
6	C85: $21^{119}-1$	84.66	29	6.0	10.0	0.1	1.0	2.5	100	70
7	C85: $76^{108}+1$	84.70	1	6.0	10.0	0.1	1.0	2.5	100	70
8	C86: $19^{153}+1$	85.24	43	6.5	10.0	0.1	1.0	2.5	100	68
9	C86: $28^{147}-1$	85.53	1	6.0	0.1	0.1	1.0	5.0	150	71
10	C87: $46^{147}-1$	86.02	1	7.0	1.0	1.0	1.0	10.0	200	74
11	C87: $74^{88}+1$	86.03	1	7.0	10.0	0.1	1.0	2.5	100	71
12	C87: $58^{73}-1$	86.07	3	6.0	0.1	0.1	1.0	5.0	150	73
13	C87: $90^{111}-1$	86.14	13	7.0	10.0	0.1	1.0	2.5	100	72
14	C87: $18^{175}+1$	86.20	19	7.0	1.0	1.0	1.0	10.0	200	76
15	C87: $52^{135}+1$	86.52	13	7.0	10.0	0.1	1.0	2.5	100	72
16	C87: $38^{108}+1$	86.72	1	6.0	0.1	0.1	1.0	5.0	150	73
17	C88: $87^{95}-1$	87.01	19	6.0	1.0	1.0	1.0	10.0	150	77
18	C88: $30^{147}+1$	87.31	43	8.0	10.0	0.1	1.0	2.5	100	73
19	C88: $77^{135}-1$	87.47	3	6.0	0.1	0.1	1.0	5.0	150	75
20	C89: $69^{135}-1$	88.19	37	7.0	0.1	0.1	1.0	5.0	150	75
21	C89: $76^{91}+1$	88.25	53	6.0	2.0	2.0	2.0	10.0	200	85
22	C89: $55^{82}+1$	88.53	13	7.0	0.1	0.1	1.0	5.0	150	75
23	C89: $91^{79}+1$	88.61	11	8.0	10.0	0.1	1.0	2.5	100	75
24	C89: $41^{91}-1$	88.64	43	6.0	1.0	1.0	1.0	2.5	200	78
25	C89: $15^{109}-1$	88.80	7	8.0	10.0	0.1	1.0	2.5	100	76
26	C89: $30^{165}-1$	88.97	1	8.0	10.0	0.1	1.0	2.5	100	75
27	C90: $30^{168}+1$	89.62	1	8.0	10.0	0.1	1.0	2.5	200	76
28	C90: $62^{117}+1$	89.86	7	8.0	10.0	0.1	1.0	2.5	200	76
29	C90: $28^{101}+1$	89.91	23	6.0	1.0	1.0	1.0	10.0	200	80
30	C91: $59^{124}+1$	90.07	1	8.0	10.0	0.1	1.0	2.5	200	75
31	C91: $28^{134}+1$	90.16	1	7.5	10.0	0.1	1.0	2.5	150	77
32	C91: $46^{117}+1$	90.47	1	8.0	10.0	0.1	1.0	2.5	200	76

Table A.3: Parameter choices and timings for numbers ranging from 85 to 96 decimal digits, factored in parallel with the self–initializing blocked variant of MPQS2 on 69 workstations. m = multiplier; $p1$ = upper bound for single large primes; $p2$ = upper bound for each of the two large primes; M = radius of the sieve interval; QT = small primes bound; RT = resulting sieve threshold. The other symbols have the same meaning as in the previous tables. We used 39 SGI Indy workstations, most of them had one R4600

#	n_c	n_1	$n_{c,1}$	n_2	$n_{c,2}$	T_s
1	7398	60896	8489	439642	*8722	292.61 h
2	22664	264667	30762	103673	11879	280.25 h
3	10396	80998	12918	566392	*1230	392.39 h
4	7590	56046	6604	276987	*10488	222.40 h
5	5652	39552	3515	487909	16457	449.67 h
6	8661	66741	9752	398599	*6196	263.47 h
7	10189	76102	11656	502131	*2778	261.54 h
8	7155	51458	5740	487829	21624	383.87 h
9	11718	85826	14421	348300	31560	457.43 h
10	16710	243584	26453	197742	23035	456.60 h
11	12593	89232	15401	436799	*201	368.59 h
12	6220	70046	10866	329402	10911	490.22 h
13	12113	87369	16487	446179	*1	458.16 h
14	15047	237174	25210	195443	21200	716.16 h
15	12469	88991	16954	571528	*1	765.22 h
16	8223	64423	8525	397334	26306	526.52 h
17	12311	215496	21763	188692	20406	842.19 h
18	11965	75465	12109	322339	*7989	348.81 h
19	5468	64788	9679	341500	10044	916.74 h
20	9435	70360	10541	399547	25169	955.96 h
21	12474	185686	14381	8149	930	1352.77 h
22	8790	66514	9770	417919	24342	770.32 h
23	15568	106549	23115	602832	*1	1250.21 h
24	9021	149997	11393	122201	7317	559.82 h
25	9132	68095	10698	431905	*12204	881.03 h
26	10081	69331	9520	403356	*12439	703.93 h
27	16667	112361	22950	545364	*1	1127.46 h
28	12997	95354	18876	426327	31156	1185.42 h
29	9752	179235	15387	204127	17769	1225.01 h
30	14859	96848	17913	370582	30980	885.15 h
31	9781	69406	9725	316247	17544	927.58 h
32	16494	105959	20861	362053	35303	961.77 h

100 MHz processor; data and instruction cache size of 16 Kb. Other machines were an SGI Power Challenge (R4400 100 MHz processor, primary cache size of 16 Kb, secondary cache size of 1 Mb), 14 HP workstations (60 MHz, negligible secondary cache) and 15 Sun (30 MHz, negligible secondary cache) workstations. An asterisk (*) means that we terminated the cycle finder as soon as we had found enough complete relations from the pp–relations to factor the number mn. Continued overleaf.

#	n	d	m	$\frac{B}{10^5}$	$\frac{L}{10^8}$	$\frac{p1}{10^8}$	$\frac{p2}{10^8}$	$\frac{M}{10^5}$	QT	RT
33	C91: $93^{97}+1$	90.94	1	7.5	10.0	0.1	1.0	2.5	100	73
34	C92: $99^{80}+1$	91.06	89	9.5	10.0	1.0	10.0	10.0	200	80
35	C92: $66^{111}+1$	91.18	61	8.0	10.0	0.1	1.0	2.5	150	80
36	C92: $98^{117}+1$	91.95	17	9.5	10.0	1.0	10.0	10.0	200	81
37	C93: $72^{134}+1$	92.25	1	8.0	10.0	0.1	1.0	2.5	150	73
38	C93: $15^{109}+1$	92.50	1	8.0	10.0	0.1	1.0	2.5	100	74
39	C93: $71^{89}+1$	92.79	1	8.0	10.0	0.1	1.0	2.5	100	75
40	C93: $54^{82}+1$	92.90	1	8.0	10.0	0.2	2.0	2.5	100	75
41	C94: $76^{86}+1$	93.17	1	8.0	10.0	0.2	2.0	2.5	100	75
42	C94: $38^{141}+1$	93.19	7	8.0	10.0	0.2	2.0	2.5	100	76
43	C94: $24^{106}+1$	93.37	1	9.0	10.0	2.0	5.0	50.0	200	83
44	C94: $61^{111}-1$	93.46	67	8.5	10.0	0.1	1.0	2.5	100	78
45	C94: $3^{570}+1$	93.55	1	8.5	15.0	0.15	1.5	6.0	150	74
46	C94: $62^{73}-1$	93.76	23	9.0	0.1	0.1	1.0	10.0	200	75
47	C95: $92^{86}+1$	94.11	1	9.5	10.0	5.0	5.0	10.0	300	79
48	C95: $43^{115}+1$	94.14	31	9.5	10.0	5.0	5.0	5.0	300	81
49	C95: $98^{91}-1$	94.17	1	9.5	10.0	5.0	5.0	5.0	200	83
50	C95: $40^{79}+1$	94.22	11	9.5	1.0	1.0	1.0	5.0	300	85
51	C95: $30^{162}+1$	94.29	1	8.0	0.1	0.1	1.0	10.0	200	79
52	C95: $41^{76}+1$	94.39	1	9.5	9.0	9.0	9.0	5.0	300	80
53	C95: $90^{86}+1$	94.44	1	9.6	5.0	5.0	5.0	5.0	300	74
54	C95: $97^{99}-1$	94.44	1	9.5	1.0	0.1	1.0	5.0	300	84
55	C95: $46^{108}+1$	94.44	1	9.6	5.0	0.1	5.0	5.0	300	96
56	C95: $62^{99}+1$	94.49	1	9.0	9.0	0.1	9.0	5.0	200	87
57	C95: $20^{121}+1$	94.57	67	8.5	0.1	0.1	1.0	10.0	200	75
58	C96: $73^{99}+1$	95.04	13	9.0	10.0	0.1	1.0	2.5	100	79
59	C96: $63^{125}-1$	95.07	59	8.5	10.0	0.1	1.0	2.5	150	77
60	C96: $84^{77}+1$	95.12	1	9.0	20.0	0.2	2.0	2.5	100	76
61	C96: $14^{140}+1$	95.14	1	9.0	20.0	0.2	2.0	2.5	100	77
62	C96: $35^{153}-1$	95.25	13	8.5	15.0	0.15	1.5	6.0	150	76
63	C96: $62^{114}+1$	95.26	5	9.0	10.0	0.15	1.5	6.0	150	75
64	C96: $75^{91}+1$	95.60	37	9.0	10.0	1.0	2.0	5.0	150	79

#	n_c	n_1	$n_{c,1}$	n_2	$n_{c,2}$	T_s
33	6629	47484	4687	530575	19905	571.87 h
34	16749	231866	22448	363895	21180	1289.41 h
35	13543	90782	16237	260008	19669	1307.58 h
36	16814	252722	25968	469502	29535	1966.29 h
37	7638	52762	5691	597520	26580	1162.23 h
38	7144	49987	5074	531343	19973	964.98 h
39	6748	48470	4760	589372	22876	1435.02 h
40	7782	72293	6369	718946	28050	1090.80 h
41	6726	63811	4888	700541	20410	1301.93 h
42	6916	75476	7796	765895	25146	1994.33 h
43	12486	243258	16636	415695	22510	3576.27 h
44	8220	57135	6802	516581	23794	1292.23 h
45	8442	66644	6463	679142	27243	1244.24 h
46	8504	57426	7401	612635	26008	2454.67 h
47	17285	397705	26619	618639	47382	2420.18 h
48	11775	288974	15323	446265	18451	2789.12 h
49	13415	317311	17401	258807	14753	2274.33 h
50	12703	202128	17455	310807	13082	2950.21 h
51	7918	58987	7127	528607	27072	3197.52 h
52	10272	286208	11208	577038	20035	2784.33 h
53	7911	202294	6932	980945	27687	2113.74 h
54	14100	86234	14416	193809	10595	3076.11 h
55	23255	116172	25304	116038	7646	2739.12 h
56	21507	123910	28258	60464	6780	2230.31 h
57	8408	57175	6878	535985	21892	3014.59 h
58	8052	56761	7703	644781	29646	2891.94 h
59	8276	57074	6939	577306	26663	2550.94 h
60	13423	117265	15710	1177729	*6813	2154.80 h
61	6835	63832	4930	823813	*23956	2420.00 h
62	6727	60373	6170	764744	25697	3356.58 h
63	7498	88908	8888	819762	26985	3160.91 h
64	7674	130350	7433	721978	28189	2959.10 h

Appendix B

The following 18 numbers 1–18 were used to determine c_1 and c_2 in approximations (4.9) and (4.11). The numbers do not interfere with those of Tables 4.4, 4.5, and 4.6. We list the square root $\sqrt{a}$ of the leading coefficient of the chosen polynomial and b. In the Tables B.1, B.2, B.3 we list the chosen parameters, the number r of sieved polynomials, α, t_1, and t_2.

Number 1 C62

$$\begin{aligned} n/m &= 11\ 9418190562\ 3383706600\ 5776102904\ 8038688463\ 4734397795\backslash \\ &\quad 0205279043 \\ \sqrt{a} &= 1137\ 7985757553 \\ b &= -122533\ 9205727893\ 5806928333 \end{aligned}$$

Number 2 C62

$$\begin{aligned} n/m &= 12\ 4960434331\ 2369771507\ 1881486773\ 5798764534\ 8378146406\backslash \\ &\quad 8316835097 \\ \sqrt{a} &= 317\ 7594878677 \\ b &= -36769\ 2049127053\ 7109069372 \end{aligned}$$

Number 3 C67

$$\begin{aligned} n/m &= 2795300\ 4544218809\ 0098428471\ 3449908979\ 6930327677\backslash \\ &\quad 1193962889\ 0773152249 \\ \sqrt{a} &= 69049\ 8676367814 \\ b &= 6141119\ 7980268423\ 3761996924 \end{aligned}$$

Number 4 C70

$$\begin{aligned} n/m &= 3452074587\ 1532036897\ 1663724885\ 9085645491\ 3776024644\backslash \\ &\quad 1573043843\ 3590687149 \\ \sqrt{a} &= 61164\ 1107106217 \\ b &= -684001154\ 8751383787\ 6228906623 \end{aligned}$$

Number 5 C71

$$\begin{aligned} n/m &= 1\ 1111111111\ 1111111111\ 1111111111\ 1111111111\ 1111111111\backslash \\ &\quad 1111111111\ 1111111111 \\ \sqrt{a} &= 59982\ 2833488191 \\ b &= 1265942212\ 7495946598\ 6178939644 \end{aligned}$$

Number 6 C73

$$\begin{aligned} n/m &= 125\ 8308688150\ 5179142348\ 1385763915\ 7371927998\ 2504149545\backslash \\ &\quad 2167806649\ 6312946019 \\ \sqrt{a} &= 495384\ 4251317617 \\ b &= 9\ 2578266926\ 6479862115\ 5975230049 \end{aligned}$$

Number 7 C73

$$\begin{aligned} n/m &= 374\ 6281167477\ 1257926128\ 5180095995\ 9020537316\ 7454521437\backslash \\ &\quad 5261287497\ 4743890821 \\ \sqrt{a} &= 351266\ 6308597937 \\ b &= 1\ 7699066017\ 4448675718\ 5749361392 \end{aligned}$$

Number 8 C76

$$\begin{aligned} n/m &= 901440\ 4467263718\ 8992383413\ 6656698645\ 6858019702\backslash \\ &\quad 1805964137\ 0498292646\ 0417171821 \\ \sqrt{a} &= 5301075\ 2767899037 \\ b &= -963\ 8357898726\ 9109565464\ 7268570179 \end{aligned}$$

Number 9 C81

$$\begin{aligned} n/m &= 1\ 2084979326\ 5793320196\ 7285818566\ 0539963519\ 8873106157\backslash \\ &\quad 9813709116\ 9898443228\ 8871633489 \\ \sqrt{a} &= 9071728\ 9289477089 \\ b &= -13159\ 6978618865\ 5546891188\ 6370038398 \end{aligned}$$

Number 10 C81

$$\begin{aligned} n/m &= 1\ 2756160780\ 8611099305\ 8711238270\ 6682428104\ 3584769443\backslash \\ &\quad 1177445428\ 2344210861\ 0440705197 \\ \sqrt{a} &= 13501794\ 5809631273 \\ b &= -55856\ 6477642561\ 2204513394\ 1344005728 \end{aligned}$$

Number 11 C81

$$\begin{aligned} n/m &= 1\ 2804626196\ 3485075791\ 3316233394\ 7732460978\ 8865639874\backslash \\ &\quad 5232331043\ 0957142640\ 4937004817 \\ \sqrt{a} &= 9145180\ 1150415749 \\ b &= 8968\ 6153120320\ 6925962274\ 5273510003 \end{aligned}$$

Number 12 C81

$$\begin{aligned} n/m &= 5\ 0637629921\ 9632522549\ 0667694055\ 0900653300\ 3228780546\backslash \\ &\quad 9758645025\ 0364355475\ 3465666097 \\ \sqrt{a} &= 25252227\ 2969049649 \\ b &= 3421\ 9602490260\ 7990556819\ 7711369718 \end{aligned}$$

Number 13 C82

$$\begin{aligned} n/m &= 54\ 1397729081\ 4966784940\ 0566002598\ 8468945940\ 6325764580\backslash \\ &\quad 6017604383\ 9674763457\ 6574474851 \\ \sqrt{a} &= 58758139\ 0028964029 \\ b &= -27533\ 1111357320\ 1577238353\ 6387564611 \end{aligned}$$

Number 14 C86

$$\begin{aligned} n/m &= 389079\ 4216696731\ 3164779897\ 1098709133\ 3423459224\backslash \\ &\quad 7603987842\ 9842778661\ 3756224933\ 3353028281 \\ \sqrt{a} &= 297038425\ 8314072489 \\ b &= -2352011\ 2511694002\ 4248915047\ 5760387015 \end{aligned}$$

Number 15 C88

$$\begin{aligned} n/m &= 11175709\ 2730877850\ 7566337550\ 9268349948\ 6353284691\backslash \\ &\quad 8694497998\ 8537535212\ 0764988950\ 2631975827 \\ \sqrt{a} &= 1760793650\ 3778546661 \\ b &= 142666646\ 7066789181\ 4484201414\ 6991688541 \end{aligned}$$

Number 16 C88

$$\begin{aligned} n/m &= 21635556\ 5990646589\ 4381332995\ 5809413444\ 1212020197\backslash \\ &\quad 1957079131\ 3709426842\ 5925590441\ 2221060057 \\ \sqrt{a} &= 1646945457\ 7510843897 \\ b &= 55968176\ 2595691602\ 3311711198\ 9664685591 \end{aligned}$$

Number 17 C91

$$
\begin{aligned}
n/m &= 1\ 3049963709\ 6764364819\ 9568631191\ 9613777969\ 9609425129\backslash \\
&\quad 6224051097\ 7317881302\ 8992552947\ 8108284379 \\
\sqrt{a} &= 8391974343\ 4446932021 \\
b &= 1799938559\ 2054466830\ 2724667288\ 2065532161
\end{aligned}
$$

Number 18 C95

$$
\begin{aligned}
n/m &= 35497\ 2695591791\ 3798837415\ 1428489573\ 4961682843\backslash \\
&\quad 8902745455\ 7880927844\ 3957786766\ 7167167895\ 2104479787 \\
\sqrt{a} &= 9\ 4010239651\ 0674712937 \\
b &= 11\ 7351838585\ 6995091351\ 3581526051\ 1685001862
\end{aligned}
$$

Number:	1	2	3	4	5	6
m	43	1	1	5	23	59
$B/10^5$	3	3.5	4	4.5	5	4.5
$L/10^6$	3	3.5	6	6.75	10	6.75
$M/10^6$	0.25	0.5	0.5	0.5	2	0.5
T	58.27	56.95	62.57	61.82	67.33	66.00
r	2475	1200	4000	9465	4964	25000
α	−2.569	−1.397	−1.629	−0.6330	−1.797	−2.035
t_1	2.947	7.389	1.963	0.5008	2.057	0.2990
t_2	12.37	29.90	10.67	3.475	13.60	1.865

Table B.1: Actual number of complete (t_1) and incomplete (t_2) relations per polynomial for the sample numbers 1–18. These data are used for the determination of the numbers c_1 and c_2 in approximations (4.9) and (4.11).

Number 1a up to Number 10a are examples used to compare the estimated and actual number of complete relations per polynomial (see Tables 4.4 and 4.5).

Number 1a C65

$$
\begin{aligned}
n/m &= 27158\ 0560707763\ 8170285090\ 4822402606\ 4919324961\backslash \\
&\quad 0446180354\ 8064706241 \\
\sqrt{a} &= 1538\ 3080127953 \\
b &= -672943\ 6594969807\ 7121164793
\end{aligned}
$$

Number:	7	8	9	10	11	12
m	5	109	1	5	1	1
$B/10^5$	5	5	6	5	4	7
$L/10^6$	5	5	180	50	200	7
$M/10^6$	0.5	0.5	2	2	2	0.5
T	65.61	71.04	81.49	76.96	79.67	73.83
r	24312	25385	69432	79328	87904	43970
α	−0.6977	−3.186	−0.9103	−1.318	−0.9949	−0.4083
t_1	0.2040	0.1916	0.1717	0.09899	0.08221	0.05388
t_2	1.251	0.9779	2.305	1.676	1.651	0.2994

Table B.2: Continuation of Table B.1.

Number:	13	14	15	16	17	18
m	11	1	43	17	19	11
$B/10^5$	7.5	8	8.5	8.5	9	8
$L/10^6$	11.25	8	8.5	8.5	9	12
$M/10^6$	1	1	1	1	1	1
T	76.43	80.01	83.51	83.38	86.58	106.1
r	13500	16068	15000	16000	27500	113000
α	−1.000	−1.251	−2.116	−1.862	−2.745	−0.8066
t_1	0.06178	0.04363	0.02687	0.02631	0.01778	0.001699
t_2	0.4461	0.2610	0.1559	0.1500	0.1044	0.0105

Table B.3: Continuation of Table B.2.

Number 2a C67

n/m = 4999228 1439980547 1200119698 6062426712 9417318768\
0936258414 0274996129

$\sqrt{a}$ = 20276 9767993829

b = −5103108 6608932988 4245660016

Number 3a C70

n/m = 3407462558 0870149333 5159834239 1305240184 2217017985\
5972598052 9382492473

$\sqrt{a}$ = 29115 0507704933

b = −200353646 9616791195 2118089548

Number 4a C70

$$\begin{aligned} n/m &= 7446263828\ 5084664090\ 2304588883\ 2293771397\ 0280869969\backslash \\ &\quad 0700499067\ 2643660783 \\ \sqrt{a} &= 91942\ 6170405581 \\ b &= -362914086\ 3289343194\ 0606822182 \end{aligned}$$

Number 5a C73

$$\begin{aligned} n/m &= 452\ 2421093387\ 6231088785\ 6962118423\ 9709711935\backslash \\ &\quad 6087407616\ 9382896713\ 5633689471 \\ \sqrt{a} &= 718570\ 3169703073 \\ b &= 4\ 9938242915\ 8672800079\ 0845023439 \end{aligned}$$

Number 6a C83

$$\begin{aligned} n/m &= 138\ 1842483120\ 1804965895\ 4919777751\ 4146329101\backslash \\ &\quad 5347888233\ 4496837656\ 5549227846\ 1653455089 \\ \sqrt{a} &= 72975948\ 9819948209 \\ b &= 194279\ 0662870352\ 6047288876\ 4531567349 \end{aligned}$$

Number 7a C85

$$\begin{aligned} n/m &= 44884\ 3150792924\ 5691032960\ 4491477299\ 0585267717\backslash \\ &\quad 7194205247\ 3840148537\ 2743386385\ 7714147711 \\ \sqrt{a} &= 339613268\ 9863474013 \\ b &= -22113030\ 0646978737\ 5654873872\ 8057541679 \end{aligned}$$

Number 8a C85

$$\begin{aligned} n/m &= 63707\ 3407732464\ 4027659439\ 0221020431\ 2154608342\backslash \\ &\quad 1294091436\ 7818192696\ 6398299023\ 2113191301 \\ \sqrt{a} &= 233339860\ 4569422437 \\ b &= -8866157\ 1010598397\ 3774112477\ 9900638768 \end{aligned}$$

Number 9a C88

$$\begin{aligned} n/m &= 74819830\ 5013400188\ 9590279365\ 2111113412\ 8394311143\backslash \\ &\quad 9946201421\ 9624982641\ 5188333977\ 2212945401 \\ \sqrt{a} &= 638716555\ 3933850141 \\ b &= 8422796\ 5151088504\ 2033019053\ 5854508860 \end{aligned}$$

Number 10a C94

n/m = 5820 8670385704 7731277020 3646825636 8097924769\
4045734980 3172591869 2043944029 4338517568 2700432109

$\sqrt{a}$ = 3 4090217783 4280797121

b = 4 0593839263 3466719333 7596276279 3179399737

Number 1b up to Number 6b are examples used to compare the estimated and actual number of incomplete relations per polynomial (see Table 4.6).

Number 1b C78

n/m = 39431014 3497913309 9512682779 8377912751 2464722783\
4573671137 3104823631 3132467031

$\sqrt{a}$ = 7812839 5401083641

b = 1623 8764058964 5821009161 0356527005

Number 2b C85

n/m = 44884 3150792924 5691032960 4491477299 0585267717\
7194205247 3840148537 2743386385 7714147711

$\sqrt{a}$ = 339613268 9863474013

b = −22113030 0646978737 5654873872 8057541679

Number 3b C85

n/m = 63707 3407732464 4027659439 0221020431 2154608342\
1294091436 7818192696 6398299023 2113191301

$\sqrt{a}$ = 233339860 4569422437

b = −8866157 1010598397 3774112477 9900638768

Number 4b C88

n/m = 74819830 5013400188 9590279365 2111113412 8394311143\
9946201421 9624982641 5188333977 2212945401

$\sqrt{a}$ = 782193202 5824032009

b = 9613471 1095933128 1036342979 5711992830

Number 5b C94

n/m = 1592 0897086948 0220278154 1767196127 6633096702\
4136090439 3052330290 7248179613 9299053383 8706754807

$\sqrt{a}$ = 1 6450910750 5595962661

b = 7895009666 7907417812 2472377015 3992652248

Number 6b C100

$$
\begin{aligned}
n/m &= 1193079720\ 2615798693\ 9665343380\ 4125664465\ 3472413608\backslash \\
&\quad 6808267215\ 2071152844\ 3608270987\ 3992085756\ 0778854537 \\
\sqrt{a} &= 22\ 1018067486\ 2769424693 \\
b &= -70\ 1459496438\ 2136923254\ 1327637222\ 2556389907
\end{aligned}
$$

Bibliography

[1] W. R. Alford and C. Pomerance. Implementing the self initializing quadratic sieve on a distributed network. In A. J. van der Poorten et al., editor, *Number–theoretic and algebraic methods in computer science*, World Scientific, pages 163–174. NTAMCS'93: Moscow, 1993, River Edge, NJ, 1995.

[2] Derek Atkins, Michael Graff, Arjen K. Lenstra, and Paul C. Leyland. THE MAGIC WORDS ARE SQUEAMISH OSSIFRAGE. In J. Pieprzyk and R. Safavi-Naini, editors, *Advances in cryptology*, volume 917 of *Lecture Notes in Computer Science*, pages 263–277. ASIACRYPT '94: Wollongong, 1994, Springer–Verlag, Berlin, etc., 1995.

[3] E. Bach and R. Peralta. Asymptotic semi–smoothness probabilities. Technical Report 1115, University of Wisconsin, Computer Sciences Department, Madison, October 1992.

[4] H. Boender. The number of relations in the quadratic sieve algorithm. Technical Report NM–R9622, Centrum voor Wiskunde en Informatica, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands, 1996. Available at `http://www.cwi.nl/cwi/publications/reports/NM-1996.html`; submitted for publication.

[5] H. Boender and H. J. J. te Riele. Factoring integers with large prime variations of the quadratic sieve. *Experimental Mathematics*, 5(4), 1996. Technical report version available at `http://www.cwi.nl/cwi/-publications/reports/NM-1995.html`.

[6] R. P. Brent and H. J. J. te Riele. Factorizations of $a^n \pm 1$, $13 \leq a < 100$. Technical Report NM–R9212, Centrum voor Wiskunde en Informatica, Kruislaan 413, 1098 SJ Amsterdam, The Netherlands, June 1992.
Updates to this report have appeared as CWI Report NM–R9419, September 1994 and CWI Report NM–9609, March 1996, with

P. L. Montgomery as coauthor. The complete tables, incorporating these updates, are available at `ftp://nimbus.anu.edu.au/pub/-Brent/factors` or `ftp://ftp.cwi.nl/pub/herman/factors`.

[7] D. M. Bressoud. *Factorization and Primality Testing.* Springer–Verlag, Berin, etc., 1989. Undergraduate Texts in Mathematics.

[8] John Brillhart, D. H. Lehmer, J. L. Selfridge, Bryant Tuckerman, and S. S. Wagstaff, Jr. Factorization of $b^n \pm 1$, $b = 2, 3, 5, 6, 7, 10, 11, 12$ up to high powers. *Contemporary Mathematics*, 22, 1988.

[9] Lucas N. H. Bunt, Phillip S. Jones, and Jack D. Bedient. *The historical roots of elementary mathematics.* Dover Publications, Inc., New York, 1988.

[10] E. R. Canfield, P. Erdős, and C. Pomerance. On a problem of Oppenheim concerning "Factorisatio Numerorum". *J. Number Theory*, 17:1–28, 1983.

[11] Henri Cohen. *A Course in Computational Algebraic Number Theory*, volume 138 of *Graduate Texts in Mathematics.* Springer–Verlag, Berlin, etc., 1993.

[12] James Cowie, Bruce Dodson, R.-Marije Elkenbracht-Huizing, Arjen K. Lenstra, Peter L. Montgomery, and Jörg Zayer. A world wide number field sieve factoring record: on to 512 bits. In K. Kim and T. Matsumoto, editors, *Advances in Cryptology*, volume 1163 of *Lecture Notes in Computer Science*, pages 382–394. ASIACRYPT '96: Kyongju, South Korea, Springer–Verlag, Berlin, etc., November 1996.

[13] A. J. C. Cunningham and H. J. Woodall. Factorisation of $y^n \mp 1$, $y = 2, 3, 5, 6, 7, 10, 11, 12$ up to high powers (n). *Hodgson*, 1925.

[14] J. A. Davis and D. B. Holdridge. Factorization using the quadratic sieve algorithm. Technical Report Sandia Report Sand 83–1346, Sandia National Laboratories, Albuquerque, New Mexico, 1983.

[15] T. F. Denny. Faktorisieren mit dem quadratischen Sieb. Master's thesis, Universität des Saarlandes, Saarbrücken, 1993.

[16] T. F. Denny, B. Dodson, A. K. Lenstra, and M. S. Manasse. On the factorization of RSA–120. In D. R. Stinson, editor, *Advances in Cryptology – CRYPTO '93*, volume 773 of *Lecture Notes in Computer Science*, pages 166–174. Springer–Verlag, Berlin, etc., 1994.

[17] Richard K. Guy. How to factor a number. In B. L. Hartnell and H. C. Williams, editors, *Congressus Numerantium XVI: Proceedings of the Fifth Manitoba Conference on Numerical Mathematics*, pages 49–89, Winnipeg, 1976. Utilitas Mathematica Publishing Incorporated.

[18] A. Hildebrand and G. Tenenbaum. On integers free of large prime factors. *Transactions of the American Mathematical Society*, 296(1):265–290, July 1986.

[19] Loo-Keng Hua. *Introduction to Number Theory.* Springer–Verlag, Berlin, etc., 1982.

[20] Donald E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming.* Addison–Wesley, Reading, MA, 2nd edition, 1981.

[21] M. Kraitchik. *Recherches sur la théorie des nombres*, volume II. Gauthier–Villar, Paris, 1929.

[22] B. A. LaMacchia and A. M. Odlyzko. Solving large sparse linear systems over finite fields. In A. J. Menezes and S. A. Vanstone, editors, *Advances in Cryptology – Proceedings of Crypto '90*, volume 537 of *Lecture Notes in Computer Science*, pages 109–133. Springer–Verlag, Berlin, etc., 1991.

[23] A. K. Lenstra, May 1996. Private communication.

[24] A. K. Lenstra and H. W. Lenstra, Jr., editors. *The development of the number field sieve*, volume 1554 of *Lecture Notes in Mathematics.* Springer–Verlag, Berlin, etc., 1993.

[25] A. K. Lenstra and M. S. Manasse. Factoring with two large primes. *Mathematics of Computation*, 63:785–798, 1994.

[26] J. van de Lune and E. Wattel. On the numerical solution of a differential–difference equation arising in analytic number theory. *Mathematics of Computation*, 23:417–421, 1969.

[27] P. L. Montgomery. Vectorization of the elliptic curve method. Manuscript. Available at `ftp://ftp.cwi.nl/pub/pmontgom/ecmvec.psl.Z`, November 1993.

[28] P. L. Montgomery. A survey of modern integer factorization algorithms. *CWI Quarterly*, 7(4):337–366, December 1994. Available at `ftp://ftp.cwi.nl/pub/pmontgom/cwisurvey.psl.Z`.

[29] P. L. Montgomery, December 1995. Private communication.

[30] P. L. Montgomery. A block Lanczos algorithm for finding dependencies over GF(2). In L. C. Guillou and J. J. Quisquater, editors, *Advances in Cryptology – Eurocrypt '95*, volume 921 of *Lecture Notes in Computer Science*, pages 106–120. Springer–Verlag, Berlin, etc., 1995. Available at `ftp://ftp.cwi.nl/pub/pmontgom/BlockLanczos.psl.Z`.

[31] M. A. Morrison and J. Brillhart. A method of factoring and the factorization of F_7. *Mathematics of Computation*, 29:183–205, 1975.

[32] K. K. Norton. *Numbers with small prime factors, and the least kth power non–residue.* AMS, Department of Mathematics, University of Michigan, Ann Arbor, Michigan 48104, 1971.

[33] Andrew M. Odlyzko. The future of integer factorization. *CryptoBytes*, 1(2):5–12, 1995. (Technical newsletter of RSA Laboratories).

[34] D. Parkinson and W. Wunderlich. A compact algorithm for gaussian elimination over GF(2) implemented on highly parallel computers. *Parallel Computing*, 1:65–73, 1984.

[35] K. Paton. An algorithm for finding a fundamental set of cycles of a graph. *Communications of the ACM*, 12:514–518, 1969.

[36] R. Peralta. A quadratic sieve on the n–dimensional hypercube. In *Advances in Cryptology*, volume 740 of *Lecture Notes in Computer Science*, pages 324–332. CRYPTO 92, 1993.

[37] C. Pomerance. The quadratic sieve factoring algorithm. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Advances in Cryptology, Proceedings of EUROCRYPT 84*, volume 209 of *Lecture Notes in Computer Science*, pages 169–182, Springer–Verlag, Berlin, etc., 1985.

[38] C. Pomerance, J. W. Smith, and R. Tuler. A pipeline architecture for factoring large integers with the quadratic sieve algorithm. *SIAM Journal on Computing*, 17:387–403, 1988.

[39] Carl Pomerance. A tale of two sieves. *Notices of the AMS*, 43(12), December 1996.

[40] H. J. J. te Riele, W. M. Lioen, and D. T. Winter. Factoring with the quadratic sieve on large vector computers. *Journal of Computational and Applied Mathematics*, 27:267–278, 1989.

[41] Herman te Riele, Walter Lioen, and Dik Winter. Factorization beyond the googol with MPQS on a single computer. *CWI Quarterly*, 4:69–72, March 1991.

[42] Hans Riesel. *Prime Numbers and Computer Methods for Factorization*, volume 57 of *Progress in Mathematics*. Birkhäuser, Boston, 1985.

[43] P. Seelhoff. Die Auflösung grosser Zahlen in ihre Factoren. *Zeitschrift für Mathematik und Physik*, 31:166–172, 1886. French translation in *Sphinx-Oedipe*, 7:84–88, 1912.

[44] R. D. Silverman. The multiple polynomial quadratic sieve. *Mathematics of Computation*, 48:329–339, 1987.

[45] R. D. Silverman. Massively distributed computing and factoring large integers. *CACM*, 34(11):95–103, November 1991.

[46] G. Wambach and H. Wettig. Block sieving algorithms. University of Cologne. Manuscript, May 1995.

[47] H. C. Williams and J. O. Shallit. Factoring integers before computers. In W. Gautschi, editor, *Mathematics of Computation 1943–1993: a Half-Century of Computational Mathematics*, volume 48, pages 481–531. Proceedings of Symposia in Applied Mathematics, American Mathematical Society, 1994.

Sources

The unblocked implementation of MPQS1, described in Chapter 2, is written by W. M. Lioen, H. J. J. te Riele, and D. T. Winter of CWI Amsterdam. The blocked implementation of MPQS1 is an extensive revision of the program of Lioen et al., written by H. Boender. Unblocked MPQS2 is written by H. Boender and it is based upon unblocked MPQS1 of Lioen et al. The blocked variant of MPQS2 is written by H. Boender and will be distributed to H. J. J. te Riele (CWI) and R. P. Brent of the Australian National University, Canberra. Contact te Riele (`herman@cwi.nl`) for information and availability of the implementations.

Chapter 3 is a revised version of [5]. Chapter 4 is a revised version of [4].

SAMENVATTING

De hoofdstelling der rekenkunde luidt:

Ieder getal groter dan 1 is te schrijven als het product van priemgetallen. Een dergelijke schrijfwijze is uniek, op de volgorde van de factoren na.

(Met "getal" bedoelen we een positief geheel getal.) Een *priemgetal* is een getal, ongelijk aan 1, dat slechts deelbaar is door 1 en zichzelf. Als een getal geen priemgetal is en niet gelijk is aan 1 dan heet het *samengesteld.* Zo is 13 een priemgetal en is 15 samengesteld, omdat $15 = 3 \times 5$. De priemgetallen die een getal delen noemen we zijn *priemfactoren.* Nu is het eenvoudig om de priemfactoren van 15 te bepalen, maar wordt het lastig om dat te doen voor

12345678912345678912345678912345678912345678 9

die er volgens de hoofdstelling der rekenkunde toch moeten zijn.

Er bestaan methoden die de priemfactoren van zeer grote getallen binnen redelijke tijd vinden mits de getallen niet al te groot zijn. (Getallen waarvan de priemfactoren eenvoudig zijn te bepalen, zoals 10^{500}, laten we buiten beschouwing.)

Het huidige record is een getal van 130 cijfers dat met behulp van honderden computers verspreid over de gehele wereld is "gekraakt". De methode die hiervoor is gebruikt heet de *number field sieve.* Deze is gebaseerd is op de *kwadratische zeefmethode*, het onderwerp van mijn onderzoek.

Twee varianten zijn MPQS1 en MPQS2 (MPQS: Multi–Polynomial Quadratic Sieve). Het is al langere tijd bekend dat MPQS2 minder tijd vergt dan MPQS1 mits de getallen "voldoende groot" zijn. Het precieze omslagpunt was tot voor kort niet bekend. Talloze voorbeeld–factorisaties hebben tot de conclusie geleid dat het omslagpunt bij 80 cijfers ligt. Met de factorisaties werden tegelijk gaten gedicht in de Cunningham tabel. Dit is een tabel van getallen van een speciale vorm waarvan men de factorisatie kent.

Bij iedere factorisatie met de kwadratische zeefmethode moet er op voorhand een aantal parameters gekozen worden. Bij een "ongelukkige" keuze is de resulterende rekentijd bijzonder groot. Zowel voor MPQS1 als voor MPQS2 hangt deze rekentijd direkt samen met het aantal zogeheten complete relaties die door de algoritmen worden gegenereerd uit zekere polynomen. Voor MPQS1 heb ik een bekende formule uitgewerkt en getest die dit aantal voorspelt. De met deze voorspellingsformule gevonden waarden wijken minder dan 10 % af van de werkelijke aantallen. Op deze manier kan men dus een goede schatting krijgen van de benodigde rekentijd van MPQS1.

Voor MPQS2 heb ik een uitdrukking gevonden om bij een vaste parameterkeuze de rekentijd te voorspellen als men getallen met een vast aantal cijfers wil factoriseren. Daarbij is slechts een klein computerexperiment nodig om tot een goede waarde van de parameters te komen. De gevonden uitdrukking is handig als men veel getallen van ongeveer dezelfde grootte wil factoriseren.

Indien men, bij een gegeven parameterkeuze, kan voorspellen hoeveel complete relaties er gegenereerd zullen worden per polynoom, dan heeft men een indruk hoe goed de parameterkeuze is. Voor MPQS2 is dat erg lastig, maar voor MPQS1 is het met behulp van analytische getaltheorie gelukt om met een redelijke nauwkeurigheid dit aantal te voorspellen. Daarbij heb ik gebruik gemaakt van een benadering van de zogenaamde psi–functie die gedefinieerd is op het eerste kwadrant van het getallenvlak. Voor bepaalde gedeelten van dit domein is bekend dat de benadering zeer goed is. Experimenteel heb ik aangetoond dat ook op sommige andere gedeelten van het getallenvlak de bekende benadering prima bruikbaar is.

Naast het wiskundig onderzoek heb ik de computerprogramma's van MPQS1 en MPQS2 geoptimaliseerd voor het gebruik op werkstations. Eerdere versies van de implementaties zijn vooral geschikt voor gebruik op een supercomputer. De nieuwe versies rekenen op werkstations ruwweg twee keer zo snel als de oude. Met de nieuwe versie van MPQS2 heb ik veel getallen gefactoriseerd voor de Cunningham tabel. Daarbij heb ik gebruik gemaakt van de faciliteiten van het Mathematisch Instituut van de Rijksuniversiteit Leiden: 69 computers zijn 2 jaar lang ingezet voor het factorisatieproject. Eén computer en enkele minuten wachttijd waren echter voldoende om het bovenstaande getal te factoriseren:

$$123456789123456789123456789123456789123456789 = 3 \times 3 \times 31 \times 41 \times 271 \times 3607 \times 3803 \times 238681 \times 2906161 \times 4185502830133110721.$$

CURRICULUM VITAE

Op 13 juli 1965 werd ik geboren in het Dijkzigt ziekenhuis te Rotterdam. Ik ben getogen in Klaaswaal (Hoeksche Waard). In Numansdorp heb ik de Christelijke mavo doorlopen en in Rotterdam de havo en het atheneum B aan de Christelijke Scholengemeenschap Johannes Calvijn.

In 1985 begon ik met de studie wiskunde aan de Rijksuniversiteit Leiden. Mijn bijvakken waren natuurkunde en informatica. De scriptie "De multi–polynomiale kwadratische zeefmethode" schreef ik samen met M. P. A. Sellink. Het onderzoek voerden we uit op het Centrum voor Wiskunde en Informatica (CWI) in Amsterdam onder leiding van dr. ir. H. J. J. te Riele. In 1991 studeerde ik af in de specialistische richting Algebra en Meetkunde bij prof. dr. R. Tijdeman.

In datzelfde jaar werd ik assistent in opleiding aan de Rijksuniversiteit Leiden met als begeleiders dr. ir. H. J. J. te Riele en prof. dr. R. Tijdeman. De resultaten van het onderzoek zijn vastgelegd in het onderhavige proefschrift. Van januari 1992 tot en met februari 1993 was ik dienstplichtig onderofficier in het Nederlandse leger.

Door de Rijksuniversiteit Leiden werd ik in staat gesteld om in oktober 1995 een werkbezoek te brengen aan dr. A. K. Lenstra van het telecommunicatiebedrijf Bellcore in Morristown (New Jersey) en in mei 1996 een congres (ANTS II) bij te wonen in Bordeaux, Frankrijk.

Vanaf 1 februari 1997 ben ik werkzaam als programmeur.

Stellingen

behorende bij het proefschrift

"Factoring Large Integers with the Quadratic Sieve"

van Hendrik Boender

1. Bij een optimale keuze van de parameters in de kwadratische zeefmethode is het aantal *direct* gevonden complete relaties per polynoom per tijdseenheid bij getallen van 94 cijfers ongeveer zeven keer zo groot als dat bij getallen van 97 cijfers.

2. Bij de keuzen van B in de kwadratische zeefmethode zoals toegepast in dit proefschrift in Appendix A, hebben polynoomwaarden $W(x)$ ongeveer een even grote kans op B–gladheid als willekeurige getallen van de grootte $W(x)/4$.

3. Als we de correctiefactor $\frac{1}{4}$ in Stelling 2 gebruiken voor een theoretische benadering van de optimale verhouding van zeeftijd en initialisatietijd, dan is de uitkomst 3.19. Dit suggereert dat de berekeningen van Alford en Pomerance [1], die uitgevoerd zijn voor willekeurige getallen en waarbij de uitkomst 2.95 is, ook toegepast kunnen worden op polynoomwaarden.

   [1] W. R. Alford en C. Pomerance. Implementing the self initializing quadratic sieve on a distributed network. In A. J. van der Poorten et al., editor, *Number–theoretic and algebraic methods in computer science*, World Scientific, blz. 163–174. NTAMCS'93: Moskou, 1993, River Edge, NJ, 1995.

4. De efficiëntie van een implementatie van de kwadratische zeefmethode op een modern werkstation hangt in belangrijke mate af van de grootte van het cache geheugen en in mindere mate van de snelheid van de processor.

5. De implementatie van een algoritme in een programmeertaal dwingt de implementator tot een ondubbelzinnige interpretatie van wat het algoritme in zijn ogen zou moeten doen.

6. Zij p_i het i–de oneven priemgetal. De gebruikelijke manier [2] om het vermoeden van Goldbach op een interval $[a, a+\delta]$, met δ veel kleiner dan a, te verifiëren is om de even getallen $n \in [a, a+\delta]$ te markeren waarvoor $n - p_1$ priem is, hetzelfde te doen voor $n - p_2, n - p_3, \ldots$, totdat *alle* even getallen gemarkeerd zijn. Het aantal niet–gemarkeerde even getallen na k van bovenstaande stappen is naar verwachting ongeveer gelijk aan

$$\frac{\delta}{2}\left(1 - 2\frac{\pi(a)}{a}\right)^k.$$

Voor $a = 4 \times 10^{11}$ en $\delta \approx 2 \times 10^6$ [2] mag men op grond hiervan verwachten gemiddeld ongeveer 180 stappen nodig te hebben. In [2] wordt wel het *maximale* (446) maar helaas niet het *gemiddelde* aantal stappen aangegeven dat nodig was voor het verifiëren van het vermoeden van Goldbach voor de even getallen $\leq 4 \times 10^{11}$. Mocht het werkelijke gemiddelde aanzienlijk afwijken van 180 stappen, dan is dat een aanwijzing dat de priemgetallen anders verdeeld zijn dan algemeen verwacht wordt.

[2] Matti K. Sinisalo, Checking the Goldbach conjecture up to 4×10^{11}, *Math. Comp.* **61** (1993), blz. 931–934.

7. Voor $q \in \mathbf{C}$ definiëren we $\mathbf{H}_q$ als de complexe unitale algebra met generatoren z, w en c en relaties $wz = qzw+(1-q)c$, $wc = cw$ en $zc = cz$. De verzamelingen $\{z^r w^s c^t \mid r, s, t \in \mathbf{Z}_{\geq 0}\}$ en $\{w^k z^l c^m \mid k, l, m \in \mathbf{Z}_{\geq 0}\}$ vormen ieder een basis van $\mathbf{H}_q$. Er geldt voor iedere $k, l, m \in \mathbf{Z}_{\geq 0}$:

$$w^k z^l c^m = \sum_{j=0}^{\min(k,l)} \gamma_j^{(k,l)}(q)\, z^{l-j} w^{k-j} c^{m+j},$$

waarbij

$$\gamma_j^{(k,l)}(q) = \frac{[k]_q!\,[l]_q!}{[k-j]_q!\,[l-j]_q!\,[j]_q!}\, q^{kl-j(k+l-1)+j(j-1)}\,(1-q)^j.$$

Hierin is

$$[k]_q = \frac{1-q^k}{1-q} \quad \text{en} \quad [k]_q! = [1]_q \cdot [2]_q \cdots [k]_q.$$

8. Zet in ieder hokje van een oneindig ruitjespapier een a of een b. Zij $P(n)$ het aantal patronen van $n \times n$ vierkanten. Als $P(n) \leq n$ voor zekere n, dan is het patroon van letters dubbelperiodiek. De bewering is niet langer waar als de bovengrens door $n + 1$ vervangen wordt. Er is een niet–periodieke invulling waarvoor $P(n) = n^2+1$ voor alle n. Als $P(2) \leq 4$, dan is de invulling periodiek.

   (Een invulling heet periodiek als de invulling invariant is onder verplaatsing over een vector die niet 0 is. Een invulling heet dubbelperiodiek als de invulling invariant is onder verplaatsing over twee lineair onafhankelijke vectoren.)

9. De moderne muziek is in sterke mate beïnvloed door zwarte Amerikaanse muziek uit de jaren '50 en '60 en Duitse experimentele en elektronische muziek uit de jaren '70.

10. Veel humor is om te huilen.