On the Number Field Sieve Integer Factorisation Algorithm

PROEFSCHRIFT

ter verkrijging van
de graad van Doctor aan de Universiteit Leiden,
op gezag van de Rector Magnificus Dr. D.D. Breimer,
hoogleraar in de faculteit der Wiskunde en
Natuurwetenschappen en die der Geneeskunde,
volgens besluit van het College voor Promoties
te verdedigen op woensdag 5 juni 2002
klokke 14.15 uur

door

Stefania Hedwig Cavallar

geboren te Meran/Merano (Italië)
in 1971

Samenstelling van de promotiecommissie:

Promotor:	
prof.dr. R. Tijdeman	
Copromotor:	
dr.ir. H.J.J. te Riele,	Centrum voor Wiskunde en Informatica
Referent:	
dr. P.L. Montgomery,	Microsoft Research USA en Centrum voor Wiskunde en Informatica
Overige leden:	
prof.dr. G. van Dijk	
prof.dr. P. Stevenhagen	
prof.dr. F. Beukers,	Universiteit Utrecht
prof.dr.ir. H.C.A. van Tilborg,	Technische Universiteit Eindhoven

THOMAS STIELTJES INSTITUTE
FOR MATHEMATICS

Chapter 3 is a slight revision of the article "Strategies in filtering in the number field sieve" which appeared in the proceedings of the ANTS IV conference [15].

Appendix A is a slight revision of the article "Factorization of RSA-140 using the number field sieve" which appeared in the proceedings of the ASIACRYPT'99 conference [16].

Appendix B is a slight revision of the article "Factorization of a 512-bit RSA modulus" which appeared in the proceedings of the EUROCRYPT 2000 conference [17].

Contents

Chapter 1

Introduction

The Number Field Sieve (NFS), since its introduction in 1988, has taken the place of the Quadratic Sieve for factoring difficult record-size integers. By difficult we mean, that methods which find small factors quickly were extensively tried on it before, but without success. Let N be the number to be factored. We distinguish between the Special and the General Number Field Sieve (SNFS and GNFS, resp.). In the first, the method is applied to integers N which can be written in the form of (what is usually called) a *special* integer, as for example $N = a^m \pm 1$. The GNFS applies to arbitrary (i.e. *general*) integers.

Like in other factorisation algorithms, the NFS constructs a few congruences $X^2 \equiv Y^2 \bmod N$. The $\gcd(X - Y, N)$ might then reveal a factor. First two monic[1] polynomials are chosen. With the help of a sieve, a large amount of integer pairs is generated for which the values of the homogenised polynomials factor over a limited set of primes. A polynomial value is the norm of an element in the number field generated by a root of the polynomial. By looking at the exponents of the first degree prime ideals in the factorisation of the elements, we select and combine useful elements and then construct pairs of squares in the number fields. The square roots in the number fields are taken and via homomorphisms these are translated into rational integer X and Y such that $X^2 \equiv Y^2 \bmod N$. We distinguish between the polynomial selection, the sieving, the filtering, the linear algebra and the square root step. In this introduction, we sketch the progress for each step made by different researchers since the appearance of [27] in 1997, thereby paying particular attention to the work described in this thesis.

1.1 Polynomial selection

The method needs two monic irreducible polynomials $f_1(x)$ and $f_2(x)$ with a common root m modulo N. For the SNFS, the simple form of N usually suggests

[1]For simplicity, we stick to monic polynomials in this short description.

suitable polynomials. For the GNFS, the common way to generate polynomials is by the base-m method. This method was improved in 1999 by Murphy and Montgomery [54]. They gave a method to generate and choose polynomial pairs which give relatively small values and at the same time have many roots modulo small primes. The method was used in the record factorisations of RSA-140 (see Appendix A) and RSA-155 (see Appendix B).

1.2 Sieving

The siever looks for *relations* which are coprime integer pairs (a, b) for which both values $f_1(a/b)b^{\deg(f_1)}$ and $f_2(a/b)b^{\deg(f_2)}$ factor over a set of small primes called the *factor base* (which are in fact the primes below the so-called *factor base bound*) while allowing a limited number of so-called *large primes* which are between the factor base bound and some *large prime bound.* The two-large-primes variant, which allows a maximum of two large primes for each polynomial, is widely used. In Chapter 2, we analyse the usefulness of allowing three large primes for one polynomial value and two large primes for the other. We compare theoretical expectations with practical results. This three-large-primes variant can be useful for sieving on computers with relatively small memory.

In 2000, Scott Contini implemented a siever into the computer algebra package Magma [44]. With this contribution, a powerful sieving program became available to a broader public.

1.3 Filtering

Imagine the following binary matrix: a column represents a relation (a, b). A typical row represents a triple (p, q, i) such that p is a prime, $f_i(q) \equiv 0 \bmod p$, $0 \leq q < p$ and for at least one relation (a, b) we have $p \mid f_i(a/b)b^{\deg(f_i)}$ with $a/b \equiv q \bmod p$. For a relation (a, b) we have an entry 1 at $(p, a/b \bmod p, i)$ if $f_i(a/b)b^{\deg(f_i)}$ is divisible by an odd power of p.

The filter step deletes useless columns and sometimes replaces two columns by their sum modulo 2 in order to 'square' certain triples. In Chapter 3, we give a description of our filter algorithm and experiences with it. We extended the original algorithm [28] with two new features. First, we additionally allow the elimination of rows that have more than three non-zero entries. Essentially, this is done by Structured Gaussian Elimination with sometimes more than one pivot relation if this gives less fill-in to the matrix. Secondly, an algorithm for removing excess columns was added. With the new filter algorithm, which was completed after the factorisation of RSA-140, we could have reduced the matrix size for RSA-140 from 4.7 million columns to 3.3 million columns. This would have led to a 33% time reduction of the linear algebra step. We successfully applied the method to the factorisation of RSA-155. The filter algorithm was integrated into Magma by Scott Contini and myself in 2000.[2]

1.4 Linear algebra

In this step, dependencies modulo 2 among the columns of the new matrix provided by the filter step have to be found. The linear algebra step is the bottleneck of the NFS because of the large amount of memory required. The Block Lanczos method by Montgomery is used [49]. For their factorisation of a 512-bit integer, Almgren et al. [2] implemented a two-processor version of Montgomery's Block Lanczos code. Montgomery also parallelised his code in 2001 and tested it with 25 parallel processors.

1.5 Square root

A dependency corresponds to a set S of relations (a, b) such that the products $\prod_{(a,b)\in S}(a-b\alpha_1)$ and $\prod_{(a,b)\in S}(a-b\alpha_2)$ are squares, say γ_1^2 and γ_2^2, in the number fields $\mathbb{Q}(\alpha_1)$ and $\mathbb{Q}(\alpha_2)$ where $\alpha_i \in \mathbb{C}$ is a root of f_i for $i = 1, 2$. By applying the homomorphisms $\phi_1 : \alpha_1 \mapsto m$ and $\phi_2 : \alpha_2 \mapsto m$ we find a congruence of two squares modulo N, namely

$$(\phi_1(\gamma_1))^2 \equiv \phi_1(\gamma_1^2) \equiv \prod_{(a,b)\in S} (a - bm) \equiv \phi_2(\gamma_2^2)) \equiv (\phi_2(\gamma_2))^2 \bmod N.$$

For a non-linear polynomial, this means we need to extract a square root of a product consisting of many terms in the respective number field. Montgomery's square root algorithm [48] was improved by Nguyen [55] in 1998.

1.6 What are the record sizes at the moment?

In the period from February 1997 to February 2002, the record size for the SNFS moved from 167 decimal digits in February 1997 to 233 digits in November 2000 [62]. For the GNFS, it moved from 130 digits achieved in April 1996 [20] to 158 digits in January 2002 [6].

[2]This part of the code is not yet released in the current version 2.8.

1.7 Notation

α	A root of the non-linear polynomial in Appendices A and B
α	A root of $f(x)$ in Chapter 3
α_i, $i = 1, 2$	A root of $f_i(x)$ in Chapter 1
α_i, $i = 1, 2$; α	$\log(B_i)/\log x$; if generic, without index, in Chapter 2
α'_i, $i = 1, 2$	$\log(B_i)/\log x'$
$\alpha(F, B)$	correction factor for polynomial values in Assumption 1
A	Bound of the sieving region (2.1)
(a, b)	Relation
β_i, $i = 1, 2$; β	$\log(L_i)/\log x$; if generic, without index
β'_i, $i = 1, 2$	$\log(L_i)/\log x'$
β	A root of g in Chapter 3
B	Bound of the sieving region (2.1)
B_i, $i = 1, 2$; B	Factor base bound for polynomial i; if generic, without index
$\mathbb{C}$	Field of complex numbers
C	Constant defined by (3.2)
$\mathrm{cont}(f)$	Content of a polynomial in $\mathbb{Z}[x]$ in Chapter 3
$\mathrm{cont}_p(r)$	Average exponent of p in factorisation of a random number r
$\mathrm{cont}_p(f)$	Average exponent of p in factorisation of a polynomial value $f(x)$
$\mathrm{cont}_p(F)$	Average exponent of p in factorisation of a homogeneous polynomial value $f(x, y)$ for $\gcd(x, y) = 1$
d_i, $i = 1, 2$	Degree of $f_i(x)$
$\deg(f)$	Degree of $f(x)$
$\mathbb{F}_2$	Field of 2 elements
$f_i(x)$, $i = 1, 2$; $f(x)$	Polynomials used for sieving; if generic, without index
$F_i(x, y)$, $i = 1, 2$; $F(x, y)$	Homogeneous polynomial of $f_i(x)$; if generic, without index
$F(x, y)$, $i = 1, 2$	Homogeneous polynomial of $f(x)$ in Chapter 3
$f(x)$	Polynomial used for sieving in Chapter 3
$G(x, y)$, $i = 1, 2$	Homogeneous polynomial of $g(x)$

	in Chapter 3
$G_0(\alpha)$	Approximation used for $\Psi(x, B)/x$
$G_i(\alpha, \beta)$	Approximation used for $\Psi_i(x, L, B)/x$
$g(x)$	Polynomial used for sieving in Chapter 3
$\gcd(x, y)$	Greatest common divisor of x and y
$H_0(x, y)$	Approximation used for $\Psi(x, B)/x$
$H_i(x, B, L)$	Approximation used for $\Psi_i(x, L, B)/x$
K	number of bits per vector element
K_1, K_2	Bounds for the $P - 1$ implementation
L_i, $i = 1, 2$; L	Large prime bound; if generic or identical for both i without index
$\mathrm{li}(x)$	Logarithmic integral
$\log y$	Natural (base e) logarithm of y
$\log_x y$	Base-x logarithm of y
M	Common root modulo N of the sieving polynomials in Chapter 3
m	Common root modulo N of the sieving polynomials
m	Dimension of the binary matrix $m \times n$ in Chapter 3
m_{max}	Maximum allowed number of relations to be added during a merge
$\mathbb{N}$	Natural numbers including 0
N	Integer being factored by NFS
n	Integer being factored by $P - 1$ in Chapter 2
n	Dimension of the binary matrix $m \times n$ in Chapter 3
ω	Weight of the binary matrix
ω_{eff}	Weight of the truncated matrix
$\omega(r)$	Weight of relation r
$\mathcal{O}(f(n, \omega))$	Any function $g(n, \omega)$ such that $g(n, \omega)/f(n, \omega)$ is bounded as $n \to \infty$ and $\omega \to \infty$
$O(f(x, \alpha, \beta, i))$	Any function $g(x, \alpha, \beta, i)$ such that $g(x, \alpha, \beta, i)/f(x, \alpha, \beta, i)$ is bounded as $x \to \infty$
$\pi(x)$	Number of primes below x
p	Usually a prime
$\Psi(x, y)$	De Bruijn's function
$\Psi_i(x, y, z)$	Generalisation of de Bruijn's function defined in Section 2.4
$\mathbb{Q}$	Field of rational numbers
$\mathbb{Q}(\alpha)$	An extension field of $\mathbb{Q}$
$\rho(x)$	Dickman's rho function
R	Sieving region, usually like (2.1)
r_i	Relation or relation-set

S	Set of dependencies
S_i, $i = 1, 2$	Sieving parameter to take account of unsieved primes or prime powers
X	Estimated number of coprime integer pairs in R
x_i, $i = 1, 2$; x	Average size of polynomial values $f_i(a, b)$ in R. Also denoted as $\overline{F}_i$
x'_i, $i = 1, 2$; x'	Size of polynomial values $f_i(a, b)$ with $\gcd(a, b) = 1$ interpreted as random numbers
$\mathbb{Z}$	Ring of integers
$\mathbb{Z}/N\mathbb{Z}$	Ring of integers modulo N
$\mathbb{Z}[x]$	Ring of polynomials with coefficients in $\mathbb{Z}$

Chapter 2

The Three-Large-Primes Variant

Abstract

The Number Field Sieve (NFS) is the asymptotically fastest known factoring algorithm for large integers. This method was proposed by John Pollard [56] in 1988. Since then several variants have been implemented with the objective of improving the siever which is the most time consuming part of this method (but fortunately, also the easiest to parallelise). Pollard's original method allowed one large prime. After that the two-large-primes variant led to substantial improvements [26]. In this chapter we investigate whether the three-large-primes variant may lead to any further improvement. We present theoretical expectations and experimental results. We assume the reader to be familiar with the NFS.

As a side-result, we improved some formulae for Taylor coefficients of Dickman's ρ function given by Patterson and Rumsey[5] and Marsaglia, Zaman and Marsaglia[45].

2.1 Introduction

In [26], B. Dodson and A. K. Lenstra describe their experiments with the two-large-primes method in the Number Field Sieve (NFS) which showed that the turnover point between the one-large-prime method and the two-large-primes method was passed for numbers ranging from 107 to 119 digits. After that, the two-large-primes method soon became widely used for larger number factorisations. In this chapter we describe experiments with the three-large-primes variant which was also employed for the special number factorisation record of 233 digits [62]. So far, the experiments do not indicate a distinct advantage over the two-large-primes version, presumably because we still have not reached the turnover point.

For sufficiently large numbers, the relations with three large primes will outnumber the relations with two large primes. But the passage from two to three large primes is not so straightforward as the passage from zero to one or from one to two large primes. Even when the three-large-primes relations outnumber the two-large-primes relations, it can still be too expensive (in time) to find sufficiently many three-large-primes relations in the sieving region. The reason for this is the rareness of successfully factored tri-composites (and with all prime factors below the large prime bound) amongst the many candidate cofactors tested.

Here we enlist a few advantages and disadvantages one expects from the three-large-primes version above the two-large-primes version. We assume that not explicitly mentioned parameters are the same in both methods. When comparing it to the two-large-primes version, one can either keep the same size of the factor base or use a smaller factor base.

If we keep the factor base the same size, we have

Advantage 1 A smaller sieving region can be taken to produce the same number of relations.

Disadvantage 1 The time needed to find a useful relation is higher. One reason is the high number of bi-composites among the candidate cofactors tested for tri-compositeness. Another reason is the larger composites which have to be factored.

Disadvantage 2 For the same number of relations, one can expect more primes to occur in the relations. It will be more difficult to combine the relations with the additional third prime to full relations. As a consequence more sieving will have to be done which might annihilate Advantage 1.

If we keep a smaller factor base, we have

Advantage 2 Less memory is needed. This is useful to sieve on machines with little memory.

Advantage 3 For a sufficiently smaller factor base, for the same number of relations, one can expect fewer primes to occur in the relations.

The two example factorisations with three large primes which we treat will show that we have not yet reached the crossover point to the three-large-primes method. For one example with 179 decimal digits ($7^{211}+1$) we kept the factor base bound artificially small (Advantage 2) to create the need of three large primes (we sieved $7^{211}-1$ with two large primes for comparison), while still not making use of all the three large primes relations since it would have been too costly in time to produce them (compared with the two-large-primes method). In this case we noticed Advantage 3.

Also for the other example, the 233-digit number $2^{773}+1$, we handled a rather small factor base bound (Advantage 2) as the number was going to be sieved in parallel on different machines and, for simplicity, we kept the same parameters for all the computers involved. Again, time considerations induced us not to detect all possible three-large-prime relations.

We did not make a comparison between the two-large-prime version and the three-large-prime version with the same factor base. This is left to further research.

In this chapter we give methods to predict the number of relations with i large primes which are found in the sieving part and compare this with real-sieved data for i at most 5. Most of these comparisons are done for numbers which were sieved with the two-large-primes method (as this is the common method at the moment) and we will see that we can reasonably well estimate the number of relations.

2.2 Outline of the chapter

In Section 2.3, we give a description of the sieving step. Here we also introduce most of the notation and terminology needed in later sections.

In Section 2.4, based on de Bruijn's Ψ function, we introduce Ψ_i to count numbers with i large primes and show a way to predict the number of smooth polynomial values in the sieving region based on heuristics originating from Peter Montgomery.

In Section 2.5, we discuss two ways to approximate Ψ by Dickman's ρ function and extend this to Ψ_i by introducing the functions G_i and H_i. We thereby generalise work done by Bach and Peralta [5] (G_1) and Lambert [37] (G_2) to three and more large primes.

In Section 2.6, we generalise (and slightly improve) a theorem by Bach and Peralta from one to two and more large primes: The Λ_i defined is an upper bound for how much the G_i approximation for Ψ_i is worse than the ρ approximation of Ψ. We measure that the Λ_is grow with i but, for $i \leq 5$ are all below 4% for the range of numbers we are interested in.

In Section 2.7, we present the numerical methods by Patterson/Rumsey and Marsaglia/Zaman/Marsaglia to compute ρ and improve upon both methods.

In Section 2.8 we present actual sieving data for two-large-primes-sieved numbers which we compare to the theoretical estimates.

In Section 2.9 we describe the obstructions which are encountered when going from two to three large primes in more detail.

In Section 2.10 we analyse data from a three-large-primes-sieved number with more details.

In Section 2.11 we compare a two-large-primes-sieved number with a three-large-primes-sieved number.

Conclusions are given in Section 2.12.

2.3 Description of the sieving step

We only describe the sieving step of the Number Field Sieve. For a complete and detailed description of the Number Field Sieve we refer to [28].

Let N be the number we want to factor. In the NFS two polynomials

$$f_j(x) = c_{j0} + c_{j1}x + \cdots + c_{jd_j}x^{d_j} \in \mathbb{Z}[x], \quad j = 1, 2,$$

are selected which are irreducible over $\mathbb{Z}$ and have a common root modulo N. We denote by

$$F_j(x,y) = f_j(x/y)y^{d_j} = c_{j0}y^{d_j} + c_{j1}xy^{d_j-1} + \cdots + c_{jd_j}x^{d_j}$$

the homogeneous form of $f_j(x)$. We call

$$R = [-A, A) \times [1, B] \bigcap \mathbb{Z} \times \mathbb{N} \tag{2.1}$$

the sieving region, where A and B are in $\mathbb{N}$. The siever looks for $(a,b) \in R$ with a and b coprime such that both $F_1(a,b)$ and $F_2(a,b)$ factor completely over the primes below the *factor base bounds* B_1 and B_2, respectively, except for at most k and l large primes which should not exceed the so-called *large prime bounds* L_1 and L_2, respectively. We call such (a,b) pairs *relations*. Following [26] we shall denote relations with k large primes in F_1 and l large primes in F_2 as k,l-*partial relations* whereas relations with no large primes are called *full relations*. We will call the method an i-large-primes variant when allowing k,l-partial relations with $\max(k,l) \le i$.[1]

We allow three large primes for the polynomial which we expect to give the larger values on the sieving region, i.e., the one with larger

$$|c_{j0}B^{d_j}| + |c_{j1}AB^{d_j-1}| + \ldots + |c_{jd_j}A^{d_j}|.$$

Let us assume in this section this is polynomial 2. Thus, our three-large-primes variant allows $2,3$-partial relations.

We sieve the roots of F_i modulo a prime p. A triple (p,q,i) denotes $0 \le q < p$ such that $F_i(q,1) \equiv 0 \bmod p$. With (p,∞,i) we denote a projective root $F_i(1,0) \equiv 0 \bmod p$ which occurs for $p \mid c_{id_i}$. These are the two different ways in which p can divide $F_i(a,b) = f_i(a/b)b^{d_i}$ with $\gcd(a,b) = 1$, namely $a/b \equiv q \bmod p$ for (p,q,i) or $p \mid b$ for (p,∞,i).

The siever sieves the triples for all the primes $p \in [d, B_i]$ where d is chosen by the user. For these triples, powers are not sieved. For triples with $p < d$ and

[1]In this respect, we differ from the notation in [26] where the method is called an i-large-primes variant, when allowing k,l-partial relations with $k + l \le i$. In this sense, the *four large primes* in the title of [26] indicate a two-large-primes variant, whereas the three-large-primes variant investigated in this chapter corresponds to five large primes in their notation. Our notation makes comparison with the Quadratic Sieve easier, where one has 1 instead of 2 polynomials. Actually, Pollard's original method is already a one-large-prime variant (in either notation), as it allows for $1,0$-partial relations.

not dividing the discriminant of the polynomial, the highest power of p below d is sieved. During the sieving process the candidate relations are marked. A relation (a, b) is considered a candidate if

$$|F_1(a,b)| \le S_1 \prod_{\substack{(p,q,1)\\ 2\le p<d\\ p|F_1(a,b)\\ p\nmid \mathrm{disc}(f_1)}} p^{\lfloor \log_p d\rfloor} \prod_{\substack{(p,q,1)\\ d\le p\le B_1\\ p|F_1(a,b)}} p \quad L_1^2$$

and

$$|F_2(a,b)| \le S_2 \prod_{\substack{(p,q,2)\\ 2\le p<d\\ p|F_2(a,b)\\ p\nmid \mathrm{disc}(f_2)}} p^{\lfloor \log_p d\rfloor} \prod_{\substack{(p,q,2)\\ d\le p\le B_2\\ p|F_2(a,b)}} p \quad L_2^3.$$

Here S_j, $j = 1, 2$, are user-chosen constants that have to take account of the primes and prime powers which are not sieved. The default value for d is 31.

The polynomials are sieved one after another. The siever we use is the so-called line-by-line siever. We give a short description for sieving polynomial 1 (2 is done analogously): the siever keeps an array for a fixed b and the single entries are indexed by a. The entries are initialised with 0. When sieving with $(p, q, 1)$, we add $\log p$ to each entry $a \equiv bq$. For a projective root $(p, \infty, 1)$, we add $\log p$ to each entry a if $p \mid b$. By storing the logarithms we can add instead of multiply the factors, which results in a time reduction even though we have to resieve afterwards. Here we used the natural logarithm, but in the implementation a more suitable base is chosen. The logarithms will be approximated.

How do we track down the candidates? We recall that (a, b) is a candidate relation if the entry with index a exceeds $\log\left(\frac{|F_1(a,b)|}{S_1 L_1^2}\right)$ and $\log\left(\frac{|F_2(a,b)|}{S_2 L_2^3}\right)$ after the sieving of polynomial 1 and 2, respectively. We divide the corresponding polynomial values by the sieved primes determined by resieving above a user-chosen threshold and by trial division below this threshold. After that we check whether the remaining parts are divisible by a higher power of those primes, or by small unsieved primes. The cofactors C_1 and C_2 do not contain any primes below B_1 and B_2, respectively.

We set a condition on the relationship between B_j and L_j. This restriction is not essential but enables us to know the maximum possible number of factors of a cofactor. We require $L_1^2 < B_1^3$ and $L_2^3 < B_2^4$. Note that it follows that $L_1 < B_1^2$, $L_2 < B_2^2$ and $L_2^2 < B_2^3$.

A cofactor C_2 of polynomial 2 is considered only if it falls into one of the three disjoint intervals $I_{21} = [B_2, L_2]$, $I_{22} = [B_2^2, L_2^2]$, $I_{23} = [B_2^3, L_2^3]$. If $C_2 \in I_{21}$ we can immediately conclude that the cofactor C_2 is prime since otherwise $C_2 \ge B_2^2 > L_2$. Similarly, if $C_2 \in I_{2i}$, then C_2 has a maximum of i primes for $i = 2, 3$. See Figure 2.1.

If a cofactor falls into I_{23}, we first perform a Rabin's probable prime test and for composites we try to find a factor with Pollard's $P - 1$ method (see section 2.3.1). If a factor smaller than L_2 is found and the remaining part

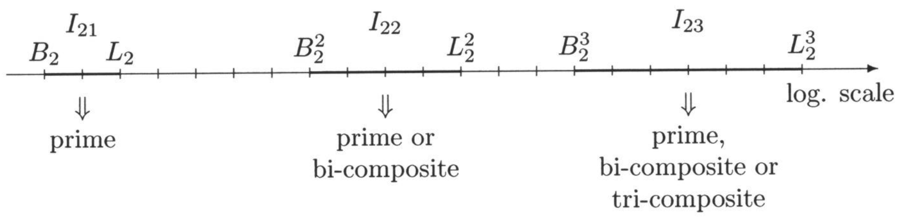

Figure 2.1:

belongs to I_{22} we proceed as for the cofactors falling into I_{22}, namely we do a probable prime test on the cofactor and, if it is composite, factor it with Shanks's SQUFOF or (if SQUFOF fails) with Pollard Rho. Then we check that the prime factors are in I_{21}.

Since the cofactors which lie centrally in the intervals are the most promising, we restrict the search to the subintervals

$$\begin{array}{c} I_{21} \\ [B_2^2, B_2^{0.1}L_2^{1.9}] \\ [B_2^{2.2}L_2^{0.8}, B_2^{1.1}L_2^{1.9}] \end{array} \tag{2.2}$$

These cut-off exponents were chosen based on few experiments. In Table 2.11 in Section 2.9 we present some experiments to measure which interval parts are more useful than others. The choice of optimal cut-off exponents however is left for further research.

The processing of C_1 is done analogously except that C_1 can only be bi-composite or prime.

The actual factorisation of cofactors for a relation is attempted only after testing size and primality of both cofactors. If there remain two composite cofactors C_1 and C_2 they are factored in the following order: the bi-composite candidate C_1 precedes a tri-composite candidate C_2. If C_2 is a bi-composite candidate, the larger of C_1 and C_2 is factored first.

2.3.1 The $P-1$ method

The $P-1$ method finds a factor p of n if $p-1$ is a product of primes below K_1 (i.e. $p-1$ is K_1-smooth) or if it is a product of primes below K_1 and one prime between K_1 and K_2 (in the terminology of Section 2.4 this means $p-1$ is $(1, K_2, K_1)$-smooth). The algorithm is split into two steps. Step 1 finds the p for which $p-1$ is K_1-smooth, step 2 the p where $p-1$ is $(1, K_2, K_1)$-smooth. In step 1, $b \equiv 2^M \bmod n$ is calculated for M the product of the prime powers below K_1. If the $\gcd(b-1, n)$ does not reveal a factor, step 2 is started using Lucas' functions. If $p-1$ is $(1, K_2, K_1)$-smooth, we will find

$\gcd(2^{Mp_i} + 2^{-Mp_i} - 2, n) > 1$ for a prime p_i in $[K_1, K_2]$. The approach with Lucas' functions is less straightforward, but nice properties of these functions allow to check many gcd's at the same time. This speeds up step 2 enormously. On the other hand, fewer gcd checks can also mean that factors are found multiplied together. In that case the algorithm starts over again, but this time with the $P+1$ method (this is repeated a few times, if necessary). The same code for Lucas' functions can be used for this. More *gcd* checks will be performed in step 1. Note that the $P+1$ method is not invoked if step 2 terminates with a gcd equal to 1. This is better for the average performance (the ratio time per found factor is smaller) as step 1 of $P+1$ is more expensive than step 1 of $P-1$. Only if $P-1$ has found the trivial factor n, we (repeatedly) try $P+1$, hoping that this method will not reduce to the $P-1$ method for all the factors.

This $P-1$ implementation does not attempt a full factorisation when there are three or more factors. Once a partial factorisation is found, SQUFOF or Pollard Rho can finish the factorisation if the cofactor is composite and in range.

The bounds K_1 and K_2 are user-chosen. The default values are $K_1 = 2\,000$ and $K_2 = 50\,000$.

This algorithm was developed and implemented by Montgomery [46].

2.4 Counting smooth numbers with Ψ

De Bruijn's function $\Psi(x, y)$ denotes the number of positive integers up to x having no prime factors larger than y. We shall call such integers y-smooth. Analogously we define, for i a positive integer and $x > y > z$, $\Psi_i(x, y, z)$ to be the number of positive integers up to x having exactly i prime factors $> z$ and $\leq y$ and the remaining prime factors $\leq z$. We shall call such integers (i, y, z)-smooth. Note that

$$\Psi_i(x, y, z) = \sum_{z<p_i\leq y} \sum_{z<p_{i-1}\leq p_i} \cdots \sum_{z<p_1\leq p_2} \Psi\left(\frac{x}{p_1\cdots p_i}, z\right). \tag{2.3}$$

We do not know simple and fast ways to calculate $\Psi(x, y)$ for large x, so we are going to use an approximation for $\Psi(x, y)$ (see Section 2.5).

2.4.1 Approximation of the number of smooth numbers among polynomial values $F(a, b)$ with $\gcd(a, b) = 1$

For our purposes, we want to approximate the number of smooth values among polynomial values given by a homogeneous polynomial in two variables $F(a, b)$ with $\gcd(a, b) = 1$.

In Subsection 2.4.2, we will compute the expected contribution to $F(a, b)$ of all primes p smaller than the factor base bound B. Therefore we calculate the average exponent of p in the factorisation of $F(a, b)$; we shall call this $\text{cont}_p(F)$. Before that, we will calculate the corresponding value for random numbers,

$\mathrm{cont}_p(r)$. We build on research by Montgomery [47], Boender [8, Chapter 4] and Murphy [54, Chapter 4].

The estimated logarithmic norm after dividing out primes in the factor base is then $\log F(a,b) - \sum_{p\le B} \mathrm{cont}_p(F)\log p$. The corresponding value for a random number y is $\log y - \sum_{p\le B}\mathrm{cont}_p(r)\log p$. According to this, we make the following

Assumption 1. *The polynomial values $F(a,b)$ are about as B-smooth as random integers with logarithmic norm $\log F(a,b) + \alpha(F,B)$ where*

$$\alpha(F,B) = \sum_{p\le B} (\mathrm{cont_p(r)} - \mathrm{cont_p(F)}) \log p.$$

According to the definition, $\Psi(x,B)/x$ gives the portion of B-smooth numbers among the numbers from 1 to x. The average size of the numbers is $(1+x)/2$ which is approximately $\overline{x} = x/2$.

The average size of $F(a,b)$ over the sieving region R (usually R is given by (2.1) is

$$\overline{F} = \frac{\iint_R |F(a,b)|\, da\, db}{\iint_R da\, db} \tag{2.4}$$

According to Assumption 1, we can treat the $F(a,b)$ values like random values of average size $\overline{x}' = \overline{F}e^{\alpha(F,B)}$, so we use $\Psi(x',B)/x'$ with $x' = 2\overline{x}'$ to approximate the portion of B-smooth polynomial values among the (a,b) pairs from R with $\gcd(a,b) = 1$.

In the sieving region we have approximately $X = \iint_R da\, db \frac{6}{\pi^2}$ pairs such that $\gcd(a,b) = 1$ (see [34, Section 4.5.2]), so we expect $X\frac{\Psi(x',B)}{x'}$ B-smooth norms among them. Because of (2.3) we can use $X\frac{\Psi_i(x',L,B)}{x'}$ as approximations for the number of (i,L,B)-smooth norms in the sieving region. If we assume that the two polynomials are independent, then we can use $X\frac{\Psi_i(x_1',L_1,B_1)}{x_1'}\frac{\Psi_j(x_2',L_2,B_2)}{x_2'}$ is an approximation of the number of i,j-partial relations. This means that we treat them as if the smoothness of $F_1(a,b)$ is unrelated to the smoothness of $F_2(a,b)$. By this we neglect some minor effect which can happen for primes which divide the resultant of the two polynomials.

2.4.2 Calculation of $\mathrm{cont}_p(r)$ and $\mathrm{cont}_p(F)$

In the sequel k shall always denote a positive integer.

Calculation of $\mathrm{cont}_p(r)$

For random numbers we expect that approximately every p^k-th number is divisible by p^k, so, taken a random number, we have probability $\frac{1}{p^k}\left(1-\frac{1}{p}\right)$ that it is divisible by p^k but not by p^{k+1}. Thus, the average exponent for p is equal to $\mathrm{cont}_p(r) = \sum_{k=1}^{\infty} \frac{k}{p^k}\left(1-\frac{1}{p}\right) = \frac{1}{p-1}$.

Calculation of $\mathrm{cont}_p(F)$

Let us first do the calculations for the simpler case of a univariate polynomial $f(x)$ which has n_{p^k} distinct roots modulo p^k. Hence a polynomial value is divisible by p^k with probability n_{p^k}/p^k. By Hensel's lemma [18], we know that if we have a non-multiple root modulo p, we have a unique corresponding non-multiple root modulo p^k. That means that for the primes which do not divide $\mathrm{disc}f$ we have $n_{p^k} = n_p$ for every $k \geq 1$ and we can easily sum the exponents of p to give $\mathrm{cont}_p(f) = \sum_{i=1}^{\infty} \frac{kn_p}{p^k}(1-\frac{1}{p}) = \frac{n_p}{p-1}$. For the primes diving $\mathrm{disc}f$ we need to calculate $\mathrm{cont}_p(f)$ manually.

Next, we look into the case of a homogeneous polynomial in two variables $F(a,b) = f(a/b)b^{\deg(f)}$. We want to estimate the probability that p^k divides $F(a,b)$ restricted to the (a,b) pairs with $\gcd(a,b) = 1$. We know that if $p^k \mid F(a,b)$, then $p^k \mid F(a+lp^k, b+mp^k)$ for l and m integers. So we can restrict our analysis to $\mathbb{Z}/p^k\mathbb{Z} \times \mathbb{Z}/p^k\mathbb{Z}$. If we take a random pair (a,b) with $\gcd(a,b) = 1$, we can reduce both a and b modulo p^k and obtain one of the $p^{2k} - p^{2k-2}$ pairs (x,y) in $\mathbb{Z}/p^k\mathbb{Z}\times\mathbb{Z}/p^k\mathbb{Z}$ with $p \nmid \gcd(x,y)$. The $p^{2(k-1)}(p^2-1)$ pairs are (almost) equally likely, because in each case $\gcd(a,b) = 1$ with probability $\frac{6}{\pi^2(1-\frac{1}{p^2})}$ if we would consider an infinite (in both dimensions) sieving region.

Consider a root a/b modulo p^k, namely $f(a/b)b^{\deg(f)} \equiv 0 \bmod p^k$. We distinguish

1. $a/b \equiv s \bmod p^k$ and $p \nmid s$. There are $\phi(p^k)$ possible s between 0 and p^k and each s has $\phi(p^k)$ different pairs $(x,y) \in \mathbb{Z}/p^k\mathbb{Z} \times \mathbb{Z}/p^k\mathbb{Z}$ with $x/y \equiv s \bmod p^k$ and $p \nmid \gcd(x,y)$.

2. $a/b \equiv s \bmod p^k$ and $p \mid s$. We have p^{k-1} possible s from 0 to p^k-1 and each s has $\phi(p^k)$ different pairs $(x,y) \in \mathbb{Z}/p^k\mathbb{Z} \times \mathbb{Z}/p^k\mathbb{Z}$ with $x/y \equiv s \bmod p^k$ and $p \nmid \gcd(x,y)$.

3. $b/a \equiv s \bmod p^k$ and $p \mid s$. This case is like 2, after exchanging x and y. These are called projective roots.

We see that whichever is the nature of a root with respect to p, we have $\phi(p^k)$ corresponding (x,y) pairs in $\mathbb{Z}/p^k\mathbb{Z}\times\mathbb{Z}/p^k\mathbb{Z}$ with $p \nmid \gcd(x,y)$. Thus, given n_{p^k} distinct roots modulo p^k (counting also possible projective roots), the probability for $F(a,b)$ with $\gcd(a,b) = 1$ to be divisible by p^k is equal to

$$\frac{n_{p^k}\phi(p^k)}{p^{2(k-1)}(p^2-1)} = \frac{n_{p^k}}{p^{k-1}(p+1)}. \tag{2.5}$$

Like in the univariate case we easily get the total $\log p$ contribution for primes not dividing $\mathrm{disc}f$: we have n_p non-multiple roots of f modulo p (counting also possible projective roots), thus we find an average exponent of p equal to

$$\mathrm{cont}_p(F) = \sum_{k=1}^{\infty} \frac{kn_p}{p^{k-1}(p+1)}\left(1-\frac{1}{p}\right) = \frac{n_p p}{p^2-1}. \tag{2.6}$$

For primes dividing disc$f$ we need to individually study the division behaviour and calculate $\mathrm{cont}_p(F)$. See Section 2.8 for information about the primes dividing the discriminant in our example polynomials.

2.5 Approximating Ψ_i

The canonical way to get an approximation for Ψ is by using Dickman's ρ function defined by

$$\rho(x) = 1 \text{ for } 0 \le x \le 1 \quad \text{and} \quad \rho'(x) = -\frac{\rho(x-1)}{x} \text{ for } x > 1. \tag{2.7}$$

By using (1.4) and (5.3) from [12], Bach and Peralta [5] deduced that if $0 < \gamma \le \alpha < 1$ and $x^\gamma \ge 2$,

$$\Psi(x, x^\alpha) = x\rho\left(\frac{1}{\alpha}\right) + O\left(\frac{x}{\gamma \log x}\right). \tag{2.8}$$

We will add the condition $\log x > \frac{1}{\alpha^2}$ which is asked for in de Bruijn's (1.4).

We have that

$$G_0(\alpha) := \lim_{x\to\infty} \frac{\Psi(x, x^\alpha)}{x} = \rho\left(\frac{1}{\alpha}\right).$$

Bach and Peralta [5] proved that, for $0 < \alpha < \beta < 1$

$$G_1(\alpha, \beta) := \lim_{x\to\infty} \frac{\Psi_1(x, x^\beta, x^\alpha)}{x} = \int_\alpha^\beta \rho\left(\frac{1-\lambda}{\alpha}\right) \frac{d\lambda}{\lambda}.$$

Lambert [37] treated the case $i = 2$: for $0 < \alpha < \beta < 1/2$,

$$G_2(\alpha, \beta) := \lim_{x\to\infty} \frac{\Psi_2(x, x^\beta, x^\alpha)}{x} = \frac{1}{2}\int_\alpha^\beta \int_\alpha^\beta \rho\left(\frac{1-(\lambda_1+\lambda_2)}{\alpha}\right) \frac{d\lambda_1}{\lambda_1}\frac{d\lambda_2}{\lambda_2}.$$

We generalise this further to: for $0 < \alpha < \beta < 1/i$ we have

$$G_i(\alpha, \beta) := \lim_{x\to\infty} \frac{\Psi_i(x, x^\beta, x^\alpha)}{x} = \tag{2.9}$$
$$\frac{1}{i!}\int_\alpha^\beta \int_\alpha^\beta \cdots \int_\alpha^\beta \rho\left(\frac{1-(\lambda_1+\lambda_2+\cdots+\lambda_i)}{\alpha}\right) \frac{d\lambda_1}{\lambda_1}\frac{d\lambda_2}{\lambda_2}\cdots\frac{d\lambda_i}{\lambda_i}.$$

In fact, in Section 2.5.1 we will prove

Theorem 1. *For a positive integer i, $0 < \alpha < \beta < 1/i$ and*

$$\log x > \frac{1}{\alpha}\max\left(\log 2, \frac{1-i\alpha}{\alpha}, \frac{1}{\log\left((i\alpha)^{-1}\right)}\right)$$

we have

$$\Psi_i(x, x^\beta, x^\alpha) =$$
$$\frac{x}{i!}\int_\alpha^\beta\int_\alpha^\beta\cdots\int_\alpha^\beta \rho\left(\frac{1-(\lambda_1+\lambda_2+\cdots+\lambda_i)}{\alpha}\right)\frac{d\lambda_1}{\lambda_1}\frac{d\lambda_2}{\lambda_2}\cdots\frac{d\lambda_i}{\lambda_i}$$
$$+O\left(\frac{\log^i((i\alpha)^{-1})}{\alpha(1-i\beta)}\frac{x}{\log x}\right).$$

The error bound is uniform in i, α and β.

By $\log^i(x)$ we mean $(\log x)^i$. Since we study $\Psi_i(x,y,z)$ in the form $\Psi_i(x,x^\beta,x^\alpha)$, we have $\alpha = \log z/\log x$ and $\beta = \log y/\log x$.

A more sophisticated approximation for $\Psi(x,y)/x$ than

$$G_0\left(\frac{\log y}{\log x}\right) = \rho\left(\frac{\log x}{\log y}\right)$$

is given by

$$H_0(x,y) = \rho\left(\frac{\log x}{\log y}\right) + \frac{1-\gamma}{\log x}\rho\left(\frac{\log x}{\log y}-1\right).$$

This approximation was used by Boender [8, Chapter 4] as well as by Murphy [54, Chapter 4] in their approximations. We define the corresponding approximations for $\Psi_i(x,y,z)$, namely

$$H_i(x,z,y) = G_i(\log_x z, \log_x y) +$$
$$\frac{1-\gamma}{\log x}\frac{1}{i!}\underbrace{\int_z^y\cdots\int_z^y}_{i \text{ times}}\rho\left(\frac{\log x - \log(t_1\cdots t_i)}{\log z}-1\right)\frac{dt_1}{t_1\log t_1}\cdots\frac{dt_i}{t_i\log t_i}. \quad (2.10)$$

with $\log x \geq \log z + i\log y$.[2]

For completeness we want to mention work by Vershik [71]. He gives formulae for calculating

$$\Phi_i(\alpha_1,\ldots,\alpha_i) = \lim_{x\to\infty} x^{-1}\left|\{n \leq x \mid p_k(n) \leq x^{\alpha_k}, k=1,\ldots,i\}\right|$$

where $1 \geq \alpha_1 \geq \cdots \geq \alpha_i \geq 0$, and $p_1(n) \geq \cdots \geq p_i(n)$ are the i largest prime divisors (counting multiplicity) of the integer n. Vershik's formulae contain (possibly different) upper bounds and no lower bound for the $p_i(n)$ values whereas our $G_i(\alpha,\beta)$ formulae contain one upper and one lower bound for the $p_i(n)$ values.

[2]or, equivalently, $\alpha + i\beta \leq 1$.

2.5.1 Proof of Theorem 1

Let us first prove some intermediate results we will use several times during the proof. We will use that

$$\pi(t) = \mathrm{li}(t) + \epsilon(t) \tag{2.11}$$

with $\mathrm{li}(t) = \int_0^t dx/\log x = \lim_{\varepsilon \to +0} \left(\int_0^{1-\varepsilon} \frac{dx}{\log x} + \int_{1+\varepsilon}^t \frac{dx}{\log x} \right)$ and

$$\epsilon(t) = O\left(\frac{t}{\log^c t}\right) \quad \text{for any} \quad c > 0. \tag{2.12}$$

Lemma 1. *For a positive integer i, $0 < \alpha < \beta < \frac{1}{i}$, $\log x > \frac{1}{\alpha \log((i\alpha)^{-1})}$ and $x^\alpha < s \le x^\beta$, we have*

$$\int_{x^\alpha}^s \frac{dt}{t \log t} < \log\left((i\alpha)^{-1}\right) \tag{2.13}$$

$$\int_{x^\alpha}^s \frac{d\epsilon(t)}{t} = O\left(\frac{1}{\alpha \log x}\right) \tag{2.14}$$

$$\sum_{x^\alpha < p \le s} \frac{1}{p} = \int_{x^\alpha}^s \frac{d\pi(t)}{t} = O\left(\log\left((i\alpha)^{-1}\right)\right). \tag{2.15}$$

Proof. For (2.13) we have

$$\begin{aligned} \int_{x^\alpha}^s \frac{dt}{t \log t} &= \log\log t\Big]_{x^\alpha}^s \le \log\log x^\beta - \log\log x^\alpha = \log\beta - \log\alpha \\ &< \log\left(\frac{1}{i}\right) - \log\alpha = \log\left((i\alpha)^{-1}\right). \end{aligned}$$

For (2.14) we integrate by parts and use (2.12) with $c = 1$ and $c = 2$ for the resulting first and second term, respectively,

$$\begin{aligned} \int_{x^\alpha}^s \frac{d\epsilon(t)}{t} &= \frac{\epsilon(t)}{t}\Big]_{x^\alpha}^s + \int_{x^\alpha}^s \frac{\epsilon(t)dt}{t^2} \\ &= O\left(\frac{1}{\log t}\Big]_{x^\alpha}^s\right) + O\left(\int_{x^\alpha}^s \frac{dt}{t\log^2 t}\right) \\ &= O\left(\frac{1}{\alpha \log x}\right). \end{aligned}$$

For (2.15) we use Stieltjes integration and find $\sum_{x^\alpha < p \le s} \frac{1}{p} = \int_{x^\alpha}^s \frac{d\pi(t)}{t}$. Then we substitute (2.11),

$$\int_{x^\alpha}^s \frac{d\pi(t)}{t} = \int_{x^\alpha}^s \frac{dt}{t \log t} + \int_{x^\alpha}^s \frac{d\epsilon(t)}{t}.$$

Thanks to (2.13) and (2.14) we can conclude the proof. □

Proof of Theorem 1.

$$\Psi_i(x, x^\beta, x^\alpha) = \sum_{x^\alpha < p_i \le x^\beta} \cdots \sum_{x^\alpha < p_1 \le p_2} \Psi\left(\frac{x}{p_1 \cdots p_i}, x^\alpha\right)$$

$$= \sum_{x^\alpha < p_i \le x^\beta} \cdots \sum_{x^\alpha < p_1 \le p_2} \Psi\left(\frac{x}{p_1 \cdots p_i}, \left(\frac{x}{p_1 \cdots p_i}\right)^{\frac{\alpha}{1-\log(p_1 \cdots p_i)/\log x}}\right) \quad (2.16)$$

We use $\alpha < \frac{\alpha}{1-\log(p_1 \cdots p_i)/\log x}$ and the theorem hypothesis in order to apply (2.8) to the latter expression. Then we divide by x which results into

$$\frac{\Psi_i(x, x^\beta, x^\alpha)}{x} = \sum_{x^\alpha < p_i \le x^\beta} \cdots \sum_{x^\alpha < p_1 \le p_2} \frac{1}{p_1 \cdots p_i} \rho\left(\frac{1 - \log(p_1 \cdots p_i)/\log x}{\alpha}\right) + O\left(\frac{1}{\alpha} \sum_{x^\alpha < p_i \le x^\beta} \cdots \sum_{x^\alpha < p_1 \le p_2} \frac{\frac{1}{p_1 \cdots p_i}}{\log(\frac{x}{p_1 \cdots p_i})}\right). \quad (2.17)$$

By $\beta < 1/i$ and (2.15), the error term from (2.17) becomes

$$O\left(\frac{1}{\alpha \log x} \sum_{x^\alpha < p_i \le x^\beta} \cdots \sum_{x^\alpha < p_1 \le p_2} \frac{1}{p_1 \cdots p_i(1 - \log(p_1 \cdots p_i)/\log x)}\right)$$

$$= O\left(\frac{1}{\alpha(1 - i\beta)\log x}\left(\sum_{x^\alpha < p \le x^\beta} \frac{1}{p}\right)^i\right) = O\left(\frac{\log^i((i\alpha)^{-1})}{\alpha(1 - i\beta)} \frac{1}{\log x}\right)$$

and by using Stieltjes integration (2.17) transforms into

$$\frac{\Psi_i(x, x^\beta, x^\alpha)}{x} = \int_{x^\alpha}^{x^\beta} \cdots \int_{x^\alpha}^{t_2} \rho\left(\frac{1 - \log(t_1 \cdots t_i)/\log(x)}{\alpha}\right) \frac{d\pi(t_1)}{t_1} \cdots \frac{d\pi(t_i)}{t_i} + O\left(\frac{\log^i((i\alpha)^{-1})}{\alpha(1 - i\beta)} \frac{1}{\log x}\right). \quad (2.18)$$

We will prove, for $x^\beta \ge \cdots \ge t_{k+1} \ge t_k \ge \cdots \ge x^\alpha$, that

$$\int_{x^\alpha}^{t_{k+1}} \cdots \int_{x^\alpha}^{t_2} \rho\left(\frac{1 - \log(t_1 \cdots t_i)/\log x}{\alpha}\right) \frac{d\pi(t_1)}{t_1} \cdots \frac{d\pi(t_k)}{t_k} = \int_{x^\alpha}^{t_{k+1}} \cdots \int_{x^\alpha}^{t_2} \rho\left(\frac{1 - \log(t_1 \cdots t_i)/\log x}{\alpha}\right) \frac{dt_1}{t_1 \log t_1} \cdots \frac{dt_k}{t_k \log t_k} + O\left(\frac{\log^{k-1}((i\alpha)^{-1})}{\alpha \log x}\right) \quad (2.19)$$

for $k \le i$ by induction on k. We define $f(t_1, \dots, t_i) = \frac{1-\log(t_1\cdots t_i)/\log x}{\alpha}$.

First we prove (2.19) for $k = 1$. We substitute (2.11), use $\rho(f(t_1, \dots, t_i)) \le 1$ and get

$$\begin{aligned}
&\int_{x^\alpha}^{t_2} \rho(f(t_1, \dots, t_i)) \frac{d\pi(t_1)}{t_1} \\
&= \int_{x^\alpha}^{t_2} \rho(f(t_1, \dots, t_i)) \frac{dt_1}{t_1 \log t_1} + \int_{x^\alpha}^{t_2} \rho(f(t_1, \dots, t_i)) \frac{d\epsilon(t_1)}{t_1} \\
&= \int_{x^\alpha}^{t_2} \rho(f(t_1, \dots, t_i)) \frac{dt_1}{t_1 \log t_1} + O(1) \int_{x^\alpha}^{t_2} \frac{d\epsilon(t_1)}{t_1}.
\end{aligned}$$

By (2.14) we can conclude the proof for $k = 1$.

Next, we assume (2.19) is true for $k < i$. Then, by the induction hypothesis, (2.11) and $\rho(f(t_1, \dots, t_i)) \le 1$,

$$\begin{aligned}
&\int_{x^\alpha}^{t_{k+2}} \cdots \int_{x^\alpha}^{t_2} \rho(f(t_1, \dots, t_i)) \frac{d\pi(t_1)}{t_1} \cdots \frac{d\pi(t_{k+1})}{t_{k+1}} \\
&= \int_{x^\alpha}^{t_{k+2}} \left(\int_{x^\alpha}^{t_{k+1}} \cdots \int_{x^\alpha}^{t_2} \rho(f(t_1, \dots, t_i)) \frac{dt_1}{t_1 \log t_1} \cdots \frac{dt_k}{t_k \log t_k} \right. \\
&\quad \left. + O\left(\frac{\log^{k-1}((i\alpha)^{-1})}{\alpha \log x} \right) \right) \frac{d\pi(t_{k+1})}{t_{k+1}} \\
&= \int_{x^\alpha}^{t_{k+2}} \cdots \int_{x^\alpha}^{t_2} \rho(f(t_1, \dots, t_i)) \frac{dt_1}{t_1 \log t_1} \cdots \frac{dt_{k+1}}{t_{k+1} \log t_{k+1}} \\
&\quad + O(1) \int_{x^\alpha}^{t_{k+2}} \int_{x^\alpha}^{t_{k+1}} \cdots \int_{x^\alpha}^{t_2} \frac{dt_1}{t_1 \log t_1} \cdots \frac{dt_k}{t_k \log t_k} \frac{d\epsilon(t_{k+1})}{t_{k+1}} \\
&\quad + O\left(\frac{\log^{k-1}((i\alpha)^{-1})}{\alpha \log x} \right) \int_{x^\alpha}^{t_{k+2}} \frac{d\pi(t_{k+1})}{t_{k+1}}.
\end{aligned}$$

We use k times (2.13) as well as (2.14) and (2.15) and find

$$O\left(\frac{\log^k((i\alpha)^{-1})}{\alpha \log x} \right)$$

as error term. This proves (2.19) for $k + 1$.

We will use (2.19) with $k = i$ and substitute x^β for t_{k+1} (this is possible, since in the last induction step we only used Lemma 1 which is also valid for $s = x^\beta$). So the first term on the right hand side of (2.18) transforms into

$$\begin{aligned}
&\int_{x^\alpha}^{x^\beta} \cdots \int_{x^\alpha}^{t_2} \rho\left(\frac{1 - \log(t_1 \cdots t_i)/\log x}{\alpha} \right) \frac{dt_1}{t_1 \log t_1} \cdots \frac{dt_k}{t_i \log t_i} \\
&\qquad + O\left(\frac{\log^{i-1}((i\alpha)^{-1})}{\alpha} \frac{1}{\log x} \right). \qquad (2.20)
\end{aligned}$$

The error term in (2.20) is contained in the error term from (2.18) since

$$\log((i\alpha)^{-1}) > \log((i\beta)^{-1}) > 1 - i\beta.$$

By symmetrising the integral bounds in (2.20) and making the substitution $\lambda_j = \log t_j / \log x$ we conclude the proof. □

2.6 Analysis of the G_i approximations

With the help of Theorem 2 we will measure how well $G_i(\alpha, \beta)$ approximates $\Psi_i(x, x^\beta, x^\alpha)/x$ under the assumption that $\rho(1/\alpha)$ is a good approximation for $\Psi(x, x^\alpha)/x$. To get an idea about the latter, see Hunter and Sorenson [31, Table 2].

Theorem 2. *Let $0 < \alpha < \beta < \frac{1}{i}$ and i be a positive integer. Assume the Riemann hypothesis. Choose c_1 and c_2 so that*

$$c_1 \le \frac{\Psi(t, t^\gamma)}{t\rho(1/\gamma)} \le c_2 \tag{2.21}$$

whenever $\frac{\alpha}{1-i\alpha} \le \gamma \le \frac{\alpha}{1-i\beta}$ and $x^{1-i\beta} \le t \le x^{1-i\alpha}$. Then, if $x^\alpha \ge 2\,657$, we have

$$c_1(1 - \Delta_{1,i}) \le \frac{\Psi_i(x, x^\beta, x^\alpha)}{xG_i(\alpha, \beta)} \le c_2(1 + \Delta_{2,i}),$$

where $\Delta_{j,i} \le \Lambda_i(\alpha, \beta, x)$ for $j = 1, 2$ and

$$\Lambda_1(\alpha, \beta, x) = \frac{\rho\left(\frac{1-\beta}{\alpha}\right)}{G_1(\alpha, \beta)} \frac{3\beta \log x}{8\pi x^{\alpha/2}} \tag{2.22}$$

and, for $i > 1$,

$$\Lambda_i(\alpha, \beta, x) = \frac{\rho\left(\frac{1-i\beta}{\alpha}\right)}{G_i(\alpha, \beta)} \left(\frac{1}{i!}\left(\left(\log\left(\frac{\beta}{\alpha}\right) + \frac{3\beta \log x}{8\pi x^{\alpha/2}}\right)^i - \log^i\left(\frac{\beta}{\alpha}\right)\right) + \left(\frac{1}{\alpha x^\alpha \log x} + \frac{\alpha \log x}{2\pi x^{3\alpha/2}}\right)\left(\log\left(\frac{\beta}{\alpha}\right) + \frac{3\beta \log x}{8\pi x^{\alpha/2}}\right)^{i-2}\right). \tag{2.23}$$

This theorem is a generalisation of Bach and Peralta's [5] Theorem 6.1 handling $i = 1$. We give a slightly better Λ_1. Table 2.1 contains values of Λ_i for $i \le 5$ for the numbers (and their parameters) we used in Sections 2.8, 2.10 and 2.11 and, in the last row, for the example from [5]. For the latter, the values of Λ_2 to Λ_5 are too large to be useful in contrast with the remaining values which are all below 4%.

In Table 2.1, $x'_k = 2\overline{F}_k e^{\alpha_k(F_k,B_k)}$ with $\overline{F}_k$ and $\alpha_k(F_k,B_k)$ defined in Section 2.4.2 and $\alpha'_k = \log_{x'_k} B_k$ and $\beta'_k = \log_{x'_k} L$ for $k = 1,2$. The entries "n.a." (not applicable) indicate that $i > 1/\beta'_k$. The calculation of the entries with "–" took too long and was abandoned.

Number	Polyn. (k)	α'_k	β'_k	x'_k	Λ_1	Λ_2	Λ_3	Λ_4	Λ_5
3,993M	1	0.186	0.205	$8.2 \cdot 10^{37}$	0.007	0.016	0.025	0.031	n.a.
3,993M	2	0.203	0.238	$5.2 \cdot 10^{32}$	0.008	0.018	0.026	0.026	n.a.
3,999L	1	0.211	0.240	$7.8 \cdot 10^{32}$	0.007	0.015	0.022	0.023	n.a.
3,999L	2	0.181	0.205	$4.3 \cdot 10^{38}$	0.007	0.015	0.024	0.030	n.a.
3,407+	1	0.169	0.190	$1.1 \cdot 10^{42}$	0.006	0.014	0.022	0.028	0.026
3,407+	2	0.211	0.241	$1.6 \cdot 10^{33}$	0.006	0.014	0.020	0.021	n.a.
3,413+	1	0.202	0.230	$6.0 \cdot 10^{34}$	0.006	0.014	0.020	0.021	n.a.
3,413+	2	0.179	0.201	$6.6 \cdot 10^{39}$	0.006	0.014	0.022	0.027	n.a.
3,427+	1	0.200	0.223	$7.2 \cdot 10^{35}$	0.006	0.013	0.020	0.021	n.a.
3,427+	2	0.179	0.198	$2.7 \cdot 10^{40}$	0.006	0.013	0.021	0.027	0.027
3,516+	1	0.204	0.234	$1.1 \cdot 10^{34}$	0.006	0.015	0.021	0.022	n.a.
3,516+	2	0.177	0.201	$3.5 \cdot 10^{39}$	0.006	0.015	0.023	0.029	n.a.
F857	1	0.166	0.188	$3.2 \cdot 10^{42}$	0.006	0.014	0.023	0.029	0.026
F857	2	0.208	0.234	$1.7 \cdot 10^{34}$	0.006	0.013	0.019	0.021	n.a.
F949	1	0.188	0.214	$2.5 \cdot 10^{37}$	0.006	0.014	0.022	–	n.a.
F949	2	0.169	0.190	$1.3 \cdot 10^{42}$	0.006	0.014	0.022	0.028	0.026
3,433+	1	0.252	0.305	$1.8 \cdot 10^{26}$	0.007	0.015	0.017	n.a.	n.a.
3,433+	2	0.154	0.170	$8.5 \cdot 10^{46}$	0.006	0.014	0.023	0.030	0.034
2,2130M	1	0.244	0.295	$1.3 \cdot 10^{27}$	0.007	0.015	0.017	n.a.	n.a.
2,2130M	2	0.152	0.168	$4.5 \cdot 10^{47}$	0.006	0.014	0.023	0.031	0.035
2,773+	1	0.161	0.198	$2.4 \cdot 10^{45}$	0.004	0.009	0.016	0.018	0.013
2,773+	2	0.162	0.200	$1.1 \cdot 10^{45}$	0.004	0.009	0.016	0.018	0.013
7, 211−	1	0.169	0.190	$3.0 \cdot 10^{42}$	0.006	0.013	0.021	0.026	0.024
7, 211−	2	0.200	0.239	$6.6 \cdot 10^{33}$	0.007	0.015	0.022	0.021	n.a.
7,211+	1	0.160	0.190	$3.0 \cdot 10^{42}$	0.007	0.017	0.028	0.034	–
7,211+	2	0.200	0.239	$6.6 \cdot 10^{33}$	0.007	0.015	0.022	0.021	n.a.
B&P	1	0.083	0.133	$1.0 \cdot 10^{60}$	0.040	0.203	0.687	1.769	3.378

Table 2.1: Λ_i, $i = 1, \ldots, 5$ for some parameter values (see Section 2.8 for the nomenclature of the numbers).

For the proof of Theorem 2 we need some estimates comparable to Lemma 1. In the following we shall use Schoenfeld's [67, Corollary 1] bound

$$|\epsilon(x)| < \frac{\sqrt{x}\log x}{8\pi} \tag{2.24}$$

which is valid under the Riemann hypothesis for $x \geq 2\,657$.

Lemma 2. *Assume the Riemann hypothesis. For $0 < \alpha < \beta$ and $x^\alpha \geq 2\,657$,* *we have*

$$\left|\int_{x^\alpha}^{x^\beta} \frac{d\epsilon(t)}{t}\right| \leq \frac{3\beta\log x}{8\pi x^{\alpha/2}} \tag{2.25}$$

$$\int_{x^\alpha}^{x^\beta} \frac{d\pi(t)}{t} \leq \log\left(\frac{\beta}{\alpha}\right) + \frac{3\beta\log x}{8\pi x^{\alpha/2}} \tag{2.26}$$

$$\int_{x^\alpha}^{x^\beta} \frac{d\pi(t)}{t^2} \leq \frac{1}{\alpha x^\alpha \log x} + \frac{\alpha\log x}{2\pi x^{3\alpha/2}} \tag{2.27}$$

Proof. For (2.25) we integrate by parts, apply (2.24) and find

$$\begin{aligned}
\left|\int_{x^\alpha}^{x^\beta} \frac{d\epsilon(t)}{t}\right| &\leq \left|\frac{\epsilon(t)}{t}\right]_{x^\alpha}^{x^\beta}\right| + \left|\int_{x^\alpha}^{x^\beta} \frac{\epsilon(t)}{t^2}dt\right| \\
&\leq \frac{|\epsilon(x^\beta)|}{x^\beta} + \frac{|\epsilon(x^\alpha)|}{x^\alpha} + \frac{1}{8\pi}\int_{x^\alpha}^{x^\beta} \frac{\log t}{t^{3/2}}dt \\
&\leq \frac{\beta\log x}{8\pi x^{\beta/2}} + \frac{\alpha\log x}{8\pi x^{\alpha/2}} + \frac{\beta\log x}{8\pi}\int_{x^\alpha}^{x^\beta} \frac{dt}{t^{3/2}} \\
&\leq \frac{3\beta\log x}{8\pi x^{\alpha/2}}.
\end{aligned}$$

For (2.26) we substitute (2.11) and use (2.25). For (2.27) we substitute (2.11), integrate by parts, apply (2.24) and use that $\frac{\log t}{t^{1/2}}$ is decreasing in the interval $[x^\alpha, x^\beta]$. We find

$$\begin{aligned}
\int_{x^\alpha}^{x^\beta} \frac{d\pi(t)}{t^2} &= \int_{x^\alpha}^{x^\beta} \frac{dt}{t^2\log t} + \frac{\epsilon(t)}{t^2}\Bigg]_{x^\alpha}^{x^\beta} + 2\int_{x^\alpha}^{x^\beta} \frac{\epsilon(t)}{t^3}dt \\
&\leq \frac{1}{\alpha\log x}\int_{x^\alpha}^{x^\beta} \frac{dt}{t^2} + \frac{|\epsilon(x^\beta)|}{x^{2\beta}} + \frac{|\epsilon(x^\alpha)|}{x^{2\alpha}} + \frac{1}{4\pi}\int_{x^\alpha}^{x^\beta} \frac{\log t}{t^{5/2}}dt \\
&\leq \frac{1}{\alpha\log x}\left(\frac{1}{x^\alpha} - \frac{1}{x^\beta}\right) + \frac{\log x^\beta}{8\pi x^{3\beta/2}} + \frac{\log x^\alpha}{8\pi x^{3\alpha/2}} + \frac{\alpha\log x}{4\pi x^{\alpha/2}}\int_{x^\alpha}^{x^\beta} \frac{dt}{t^2} \\
&\leq \frac{1}{\alpha x^\alpha \log x} + \frac{\alpha\log x}{2\pi x^{3\alpha/2}}.
\end{aligned}$$

□

Proof of Theorem 2. We start from equation (2.16) and symmetrise the summation bounds. In the first term on the right hand side of (2.28) we miscount the cases that some of the $p_1, \ldots, p_i$ are equal. The second term is a large upper bound for the correction.

$$\begin{aligned}\Psi_i(x, x^\beta, x^\alpha) \leq &\\ \frac{1}{i!} \sum_{x^\alpha < p_i \leq x^\beta} \cdots \sum_{x^\alpha < p_1 \leq x^\beta} &\Psi\left(\frac{x}{p_1 \cdots p_i}, \left(\frac{x}{p_1 \cdots p_i}\right)^{\frac{\alpha}{1-\log(p_1\cdots p_i)/\log x}}\right) + \\ &\sum_{x^\alpha < p_{i-1} \leq x^\beta} \cdots \sum_{x^\alpha < p_1 \leq x^\beta} \Psi\left(\frac{x}{p_1 \cdots p_{i-2} p_{i-1}^2},\right. \\ &\left.\left(\frac{x}{p_1 \cdots p_{i-2} p_{i-1}^2}\right)^{\frac{\alpha}{1-\log(p_1\cdots p_{i-2}p_{i-1}^2)/\log x}}\right). \qquad (2.28)\end{aligned}$$

By using Stieltjes integration and from the assumption (2.21) we get

$$\begin{aligned}\frac{\Psi_i(x, x^\beta, x^\alpha)}{x} \leq &\\ c_2 \Bigg(\frac{1}{i!} &\underbrace{\int_{x^\alpha}^{x^\beta} \cdots \int_{x^\alpha}^{x^\beta}}_{i \text{ times}} \rho\left(\frac{1 - \log(t_1 \ldots t_i)/\log x}{\alpha}\right) \frac{d\pi(t_1)}{t_1} \cdots \frac{d\pi(t_i)}{t_i} + \\ &\underbrace{\int_{x^\alpha}^{x^\beta} \cdots \int_{x^\alpha}^{x^\beta}}_{i-1 \text{ times}} \rho\left(\frac{1 - \log(t_1 \ldots t_{i-2} t_{i-1}^2)/\log x}{\alpha}\right) \frac{d\pi(t_1)}{t_1} \cdots \\ &\cdots \frac{d\pi(t_{i-2})}{t_{i-2}} \frac{d\pi(t_{i-1})}{t_{i-1}^2}\Bigg). \qquad (2.29)\end{aligned}$$

We substitute (2.11) for the first part and for the second part we bound

$$\rho\left(\frac{1 - \log(t_1 \ldots t_{i-2} t_{i-1}^2)/\log x}{\alpha}\right)$$

by $\rho\left(\frac{1-i\beta}{\alpha}\right)$, apply (2.27) as well as $i - 2$ times (2.26) and get

$$\begin{aligned}\frac{\Psi_i(x, x^\beta, x^\alpha)}{x} \leq c_2 \Bigg(&G_i(\alpha, \beta) + \frac{1}{i!} \sum_{j=1}^{i} \binom{i}{j} E_{i-j,j} + \\ &\rho\left(\frac{1 - i\beta}{\alpha}\right) \left(\frac{1}{\alpha x^\alpha \log x} + \frac{\alpha \log x}{2\pi x^{3\alpha/2}}\right) \cdot \\ &\sum_{j=0}^{i-2} \binom{i-2}{j} \log^{i-2-j}\left(\frac{\beta}{\alpha}\right) \left(\frac{3\beta \log x}{8\pi x^{\alpha/2}}\right)^j\Bigg) \qquad (2.30)\end{aligned}$$

where

$$E_{i-j,j} = \int_{x^\alpha}^{x^\beta} \cdots \int_{x^\alpha}^{x^\beta} \rho\left(\frac{1-\log(t_1 \ldots t_i)/\log x}{\alpha}\right) \frac{dt_1}{t_1 \log t_1} \cdots \frac{dt_{i-j}}{t_{i-j} \log t_{i-j}} \frac{d\epsilon(t_{i-j+1})}{t_{i-j+1}} \cdots \frac{d\epsilon(t_i)}{t_i}. \tag{2.31}$$

In order to find an upper bound for $E_{i-j,j}$ we bound ρ in (2.31) by $\rho\left(\frac{1-i\beta}{\alpha}\right)$, apply $i-j$ times $\int_{x^\alpha}^{x^\beta} \frac{dt}{t \log t} = \log\left(\frac{\beta}{\alpha}\right)$ and j times (2.25) and obtain

$$E_{i-j,j} \le \rho\left(\frac{1-i\beta}{\alpha}\right) \log^{i-j}\left(\frac{\beta}{\alpha}\right) \left(\frac{3\beta \log x}{8\pi x^{\alpha/2}}\right)^j. \tag{2.32}$$

We use (2.32) to bound the right hand side of (2.30). We divide both sides of the new inequality by $G_i(\alpha,\beta)$ to get

$$\frac{\Psi_i(x, x^\beta, x^\beta)}{x G_i(\alpha,\beta)} \le c_2 (1 + \Lambda_i)$$

with Λ_i given by (2.22) and (2.23), respectively.

The lower bound is proven by an entirely analogous argument, starting from the left hand side inequality in (2.21). Actually, the proof is simpler for the left hand side since we do not need a correction term as given by the second term on the right hand side of (2.29). □

2.7 Calculating ρ

We want to calculate Dickman's ρ function to high precision, as our estimates of smooth numbers rely on values of the ρ function.

The ρ function is the solution of a so-called differential-difference equation. This implies that it is piecewise analytic. For example

$$\rho(x) = \begin{cases} 1 & \text{if } 0 \le x \le 1 \\ 1 - \log x & \text{if } 1 \le x \le 2 \end{cases}$$

For all the other intervals of length 1 we can write ρ as a Taylor series where the coefficients depend on the Taylor series coefficients of ρ in the left adjacent interval. In order to guarantee correct results up to a certain precision, we use two methods. One was used first by Bach and Peralta [5] and is due to Patterson and Rumsey; it expands the series on the right end of the intervals. The other method is by Marsaglia, Zaman and Marsaglia [45] and expands the series around the midpoints of the intervals.[3] Although M&Z&M give nearly

[3]One might also think of expanding the series on the left end of the intervals, i.e. writing $\rho(k+\xi) = \sum_i^\infty c_i^{(k)} \xi^i$ for $0 \le \xi \le 1$. The formulae can be got analogously and would be even simpler as in the other two cases, but the method is impractical as too many terms are needed due to the very slow convergence of the sums. For example, $c_i^{(1)} = \frac{(-1)^i}{i}$.

correct values for $\rho(10)$, $\rho(15)$ and $\rho(20)$, there are a few oversights in their formulae. We provide the correct relations in Table 2.2.

Patterson & Rumsey	Marsaglia & Zaman & Marsaglia
$0 \le \xi \le 1$	$-1 \le \xi \le 1$
$\rho(k-\xi) = \sum_{i=0}^{\infty} c_i^{(k)} \xi^i$	$\rho\left(k+\frac{1}{2}+\frac{1}{2}\xi\right) = \sum_{i=0}^{\infty} c_i^{(k)} \xi^i$
$k = 1, 2, \ldots$	$k = 0, 1, \ldots$
$c_0^{(1)} = 1$	$c_0^{(0)} = 1$
$c_i^{(1)} = 0 \quad \text{for} \quad i > 0$	$c_i^{(0)} = 0 \quad \text{for} \quad i > 0$
$c_0^{(2)} = 1 - \log 2$	$c_0^{(1)} = 1 - \log(\frac{3}{2})$
$c_i^{(2)} = \frac{1}{i2^i} \quad \text{for} \quad i > 0$	$c_i^{(1)} = \frac{(-1)^i}{i3^i} \quad \text{for} \quad i > 0$
$c_i^{(k)} = \frac{1}{i} \sum_{j=0}^{i-1} \frac{c_j^{(k-1)}}{k^{i-j}}$ for $i > 0$	$c_i^{(k)} = -\frac{c_{i-1}^{(k-1)}+(i-1)c_{i-1}^{(k)}}{i(2k+1)}$ for $i > 0$
$c_0^{(k)} = \frac{1}{k-1} \sum_{j=1}^{\infty} \frac{c_j^{(k)}}{j+1}$	$c_0^{(k)} = \sum_{j=0}^{\infty} \left(c_j^{(k-1)} + \frac{(-1)^{j+1} m_j^{(k-1)}}{j+1}\right)$
for $k > 1$	$m_0^{(k)} = \frac{c_0^{(k)}}{2k+3}$
	$m_i^{(k)} = \frac{c_i^{(k)} - m_{i-1}^{(k)}}{2k+3} \quad \text{for} \quad i > 0$
	for $k > 0$

Table 2.2: Taylor coefficients for ρ - original methods

Bach and Peralta found that for computing $\rho(x)$ in the range $0 \le x \le 20$ with a relative error of about 10^{-17}, it is sufficient to approximate the infinite sums in P&R's method with the sums of the first 55 terms. Table 2.3 reproduces the number of terms and the working precision required to guarantee 16, 32 or 64 correct digits of ρ in the range $0 \le x \le 20$ for each of the two methods. The calculations were done with MATHEMATICA's arbitrary precision: MATHEMATICA maintains as much precision as possible and, if necessary, performs internal intermediate calculations to up to 50 more digits. If it loses precision because of roundoff errors, only the correct digits are returned. In Table 2.3 we also reproduce the time needed to calculate all the c's on an SGI O2 MIPS R5000 180 MHz. In order to determine how many terms and how many digits of working precision were needed, we first chose sufficiently high values for both methods to have the results $\rho(i)$ for $i = 1, \ldots, 20$ coincide to a high number of digits for the two methods. We took that as a reference solution. Then we varied MATHEMATICA's working precision to find the lower bound for the working precision and after that we determined the minimum number of terms.

	Patterson & Rumsey			Marsaglia & Zaman & Marsaglia		
order rel. error	number of terms	working precision	time in s	number of terms	working precision	time in s
10^{-17}	42	17	13	87	45	6
10^{-33}	91	32	70	120	61	9
10^{-65}	195	64	400	188	93	15

Table 2.3: Original methods

Both methods have some drawback. P&R have to calculate a sum of i terms for each $c_i^{(k)}(i > 0, k > 2)$ which becomes costly in time when we need high precision and thus many terms. M&Z&M on the other hand have a very involved way to calculate $c_0^{(k)}(k > 1)$. Their treatment requires many more digits of precision than P&R.

We were able to simplify both P&R's $c_i^{(k)}(i > 0, k > 2)$ and M&Z&M's $c_0^{(k)}(k > 1)$ so they resemble the corresponding c in the other method. Therefore we only needed to follow the approach taken in the other method. For the derivation of $c_0^{(k)}$ this means to use $\rho(x) = \frac{1}{x}\int_{x-1}^{x} \rho(t)dt$ for $x > 1$ instead of the integral form of (2.7) which was used by M&Z&M. For M&Z&M, this avoids the need of developing $\frac{\rho(k-1/2+\xi/2)}{2k+1+\xi}$ into power series which leads to the auxiliary coefficients m_i. For the derivation of $c_i^{(k)}(i > 0)$ we used (2.7). Then, for P&R, we did NOT develop $1/(1 - \xi/k)$ as a power series as described in [4]. For the new proofs, see Appendix 2.A.

In Table 2.4 we give the simplified terms together with the similar term in the other method. The recursive formula for P&R's $c_i^{(k)}(i > 1, k > 2)$ can

Patterson & Rumsey	Marsaglia & Zaman & Marsaglia
$c_i^{(k)} = \frac{c_{i-1}^{(k-1)}+(i-1)c_{i-1}^{(k)}}{ik} \quad i > 0$	$c_i^{(k)} = -\frac{c_{i-1}^{(k-1)}+(i-1)c_{i-1}^{(k)}}{i(2k+1)} \quad i > 0$
$c_0^{(k)} = \frac{1}{k-1}\sum_{j=1}^{\infty} \frac{c_j^{(k)}}{j+1}$	$c_0^{(k)} = \frac{1}{2k}\left(c_0^{(k-1)} + \sum_{j=1}^{\infty} \frac{c_j^{(k-1)}+(-1)^j c_j^{(k)}}{j+1}\right)$
for $k > 1$	for $k > 0$

Table 2.4: New form of Taylor coefficients for ρ compared to the corresponding term in the other method

also easily be derived from

$$\begin{aligned} c_i^{(k)} &= \frac{1}{i}\sum_{j=0}^{i-1}\frac{c_j^{(k-1)}}{k^{i-j}} = \frac{1}{ik}\sum_{j=0}^{i-1}\frac{c_j^{(k-1)}}{k^{i-1-j}} = \frac{1}{ik}\left(\sum_{j=0}^{i-2}\frac{c_j^{(k-1)}}{k^{i-1-j}} + c_{i-1}^{(k-1)}\right) \\ &= \frac{1}{ik}\left((i-1)c_{i-1}^{(k)} + c_{i-1}^{(k-1)}\right). \end{aligned}$$

With the new recursion formula P&R has become 6 to 44 times faster. The simplified M&Z&M needs 20 fewer digits of precision and between 56 and 58 fewer terms to produce the same relative error as the original version. Moreover, it became 2 to 3 times faster. See Table 2.3 and 2.5 for some numerical data.

	Patterson & Rumsey			Marsaglia & Zaman & Marsaglia		
order rel. error	number of terms	working precision	time in s	number of terms	working precision	time in s
10^{-17}	42	17	2	31	25	2
10^{-33}	91	32	4	64	41	3
10^{-65}	195	64	9	130	73	7

Table 2.5: Improved methods

A further slight improvement in time can be achieved by also using the recursive forms for P&R's $c_i^{(2)}$ and M&Z&M's $c_i^{(1)}$ $(i > 0)$.

Note that in order not to bump into MATHEMATICA's recursion limit one should give P&R's $c_1^{(k)}$ $(k > 2)$ and M&Z&M's $c_1^{(k)}$ $(k > 1)$ explicitly.

2.8 Examples

In this section we compare real sieve data with the approximations we derived in 2.4.1 for several example numbers. We tried to have some variety in our examples by including Cunningham numbers as well as Fibonacci numbers. All the numbers were factored by Montgomery at the time of our experiments. In Tables 2.7 and 2.8 we reproduce the parameters used in the factorisations.

With $x, y+$ we denote $x^y + 1$. With $2, 2h$M we denote the Aurifeuillian factor [11, III.C.2] $2^h + 2^{\frac{h+1}{2}} + 1$ of $2, 2h+$. Similarly, $3, 3h$M is short for $3^h + 3^{\frac{h+1}{2}} + 1$ and $3, 3h$L for $3^h - 3^{\frac{h+1}{2}} + 1$. The numbers Fx are Fibonacci numbers, the numbers Lx Lucas numbers.

The numbers were sieved with the Special Number Field Sieve (SNFS), except for 2,2130M and 3,433+, which were sieved with the General Number Field Sieve (GNFS). Also for the latter two we write the numbers to be factored as algebraic factors, even though we are actually factoring a cofactor. The cofactor sizes are stated in Tables 2.7 and 2.8. The SNFS difficulty is given by the resultant of the polynomials.

3,993M	$f_1(x) = 3^{55}x - 1$	
	$f_2(x) = x^6 + 3x^3 + 3$	$\mathrm{cont}_3(F_2) = \frac{1}{4}$
3,999L	$f_1(x) = 3^{55}x - 1$	
	$f_2(x) = x^6 - 9x^3 + 27$	$\mathrm{cont}_3(F_2) = \frac{3}{4}$
3,407+	$f_1(x) = 3^{37}x - 3^{74} - 1$	
	$f_2(x) = x^5 - x^4 - 4x^3 + 3x^2 + 3x - 1$	$\mathrm{cont}_{11}(F_2) = \frac{1}{12}$
3,413+	$f_1(x) = x - 3^{59}$	
	$f_2(x) = x^6 - x^5 + x^4 - x^3 + x^2 - x + 1$	$\mathrm{cont}_7(F_2) = \frac{1}{8}$
3,427+	$f_1(x) = x - 3^{61}$	
	$f_2(x) = x^6 - x^5 + x^4 - x^3 + x^2 - x + 1$	$\mathrm{cont}_7(F_2) = \frac{1}{8}$
3,516+	$f_1(x) = 3^{57}x - 1$	
	$f_2(x) = x^6 + 3x^3 + 9$	$\mathrm{cont}_3(F_2) = \frac{1}{2}$
F857	$f_1(x) = \mathrm{F171}x - \mathrm{F172}$	
	$f_2(x) = x^5 + 5x^4 + 10x^2 - 5x + 2$	$\mathrm{cont}_5(F_2) = \frac{1}{3}$
F949	$f_1(x) = x - \mathrm{L146}$	
	$f_2(x) = x^6 - x^5 - 5x^4 + 4x^3 + 6x^2 - 3x - 1$	$\mathrm{cont}_{13}(F_2) = \frac{1}{14}$
3,433+	$f_1(x) = x - 1018022109428884191058$	
	$f_2(x) = 5821578000x^5$	$\mathrm{cont}_2(F_2) = 3$
	$-13767381653260x^4$	$\mathrm{cont}_3(F_2) = \frac{9}{8}$
	$-3504111252981476x^3$	$\mathrm{cont}_5(F_2) = \frac{25}{24}$
	$+5033731003610092975x^2$	$\mathrm{cont}_7(F_2) = \frac{13}{48}$
	$+41414643218036780062x$	$\mathrm{cont}_{61}(F_2) = \frac{1}{62}$
	$-563572130841392284366681$	$\mathrm{cont}_{881}(F_2) = \frac{1}{882}$
2,2130M	$f_1(x) = x - 5310903123331135610192$	
	$f_2(x) = 6590263680x^5$	$\mathrm{cont}_2(F_2) = \frac{8}{3}$
	$-71058983292296x^4$	$\mathrm{cont}_3(F_2) = \frac{3}{2}$
	$+10126751094225398x^3$	$\mathrm{cont}_5(F_2) = \frac{7}{12}$
	$+349867764197537945x^2$	$\mathrm{cont}_{19}(F_2) = \frac{37}{360}$
	$-5404582433335517396810x$	$\mathrm{cont}_{41}(F_2) = \frac{27}{560}$
	$+2581409262310033997312415$	$\mathrm{cont}_{2003}(F_2) = \frac{8011}{4012008}$

Table 2.6: Detailed polynomials data

name	3,993M	3,999L	3,407+	3,413+	3,427+
SNFS difficulty	158	159	177	169	175
cofactor size	144	149	148	135	169
degree $f_1(x)$	1	1	1	1	1
degree $f_2(x)$	6	6	5	6	6
A	1680000	2520000	3600000	3360000	4200000
B	1560000	1250000	3000000	2400000	3200000
X	$3.18651 \cdot 10^{12}$	$3.82999 \cdot 10^{12}$	$1.31312 \cdot 10^{13}$	$9.80474 \cdot 10^{12}$	$1.63411 \cdot 10^{13}$
B_1	4400000	8500000	13000000	11000000	14500000
B_2	11000000	10000000	10000000	13000000	17000000
L	60000000	80000000	100000000	100000000	100000000
S_1	15	30	40	30	30
S_2	60	30	7	7	7
x_1	$2.93074 \cdot 10^{32}$	$4.39611 \cdot 10^{32}$	$6.08267 \cdot 10^{41}$	$3.39129 \cdot 10^{34}$	$4.06955 \cdot 10^{35}$
x_2	$1.87775 \cdot 10^{37}$	$1.02598 \cdot 10^{38}$	$1.53030 \cdot 10^{32}$	$6.13546 \cdot 10^{38}$	$2.54655 \cdot 10^{39}$
$\alpha(F_1, B_1)$	0.569915	0.569915	0.569915	0.569915	0.569915
$\alpha(F_2, B_2)$	1.468072	1.429203	2.319329	2.378699	2.377064
full relations	297961/0.56/0.65	412555/0.54/0.62	387672/0.63/0.73	502027/0.53/0.61	684987/0.55/0.63
0,1-partial rels.	481365/0.61/0.69	873553/0.58/0.66	737783/0.67/0.76	1047129/0.58/0.65	1205720/0.59/0.67
0,2-partial rels.	268380/0.68/0.76	633695/0.66/0.73	446398/0.74/0.81	759311/0.64/0.72	741788/0.66/0.73
1,0-partial rels.	769170/0.57/0.65	806649/0.55/0.62	944266/0.64/0.72	1008690/0.54/0.61	1194986/0.56/0.63
1,1-partial rels.	1248973/0.62/0.69	1711506/0.59/0.66	1799413/0.68/0.75	2116479/0.59/0.65	2107447/0.60/0.66
1,2-partial rels.	694993/0.70/0.76	1245009/0.67/0.73	1085377/0.74/0.81	1532260/0.66/0.71	1299863/0.67/0.72
2,0-partial rels.	627188/0.61/0.68	500656/0.58/0.64	819125/0.66/0.74	655488/0.58/0.64	686676/0.59/0.65
2,1-partial rels.	1018741/0.66/0.72	1065195/0.62/0.68	1565368/0.70/0.77	1374882/0.62/0.68	1217910/0.63/0.68
2,2-partial rels.	568849/0.74/0.79	780025/0.70/0.74	946628/0.77/0.82	1003843/0.69/0.74	752013/0.70/0.75
total relations	5975620/0.64/0.71	8028843/0.61/0.68	8732030/0.69/0.77	10000109/0.61/0.67	9891390/0.61/0.68
1.6li(L)	5700294	7472145	9219535	9219535	9219535
sieving time (days)	148	131	98	59	112

Table 2.7: Examples

name	3,516+	F857	F949	3,433+	2,2130M
SNFS difficulty	165	179	184	n.a.	n.a.
cofactor size	161	179	157	115	118
degree $f_1(x)$	1	1	1	1	1
degree $f_2(x)$	6	5	6	5	6
A	3900000	6000000	8400000	70200000	97200000
B	1600000	3050000	4400000	100000	135000
X	$7.58694 \cdot 10^{12}$	$2.22502 \cdot 10^{13}$	$4.49380 \cdot 10^{13}$	$8.53532 \cdot 10^{12}$	$1.59545 \cdot 10^{13}$
B_1	8500000	11000000	11000000	4200000	4200000
B_2	10000000	13000000	13000000	16777215	16777215
L	90000000	100000000	100000000	100000000	100000000
S_1	20	30	30	20	40
S_2	30	25	1	100	2000
x_1	$6.12316 \cdot 10^{33}$	$1.79418 \cdot 10^{42}$	$1.43103 \cdot 10^{37}$	$1.01802 \cdot 10^{26}$	$7.16972 \cdot 10^{26}$
x_2	$1.04850 \cdot 10^{39}$	$6.25192 \cdot 10^{33}$	$5.73419 \cdot 10^{40}$	$4.87448 \cdot 10^{48}$	$1.65831 \cdot 10^{50}$
$\alpha(F_1, B_1)$	0.569915	0.569915	0.569915	0.569915	0.569915
$\alpha(F_2, B_2)$	1.193893	1.002230	3.153286	−4.046483	−5.915719
full relations	408537/0.47/0.54	393668/0.59/0.68	359222/0.46/0.53	446527/0.44/0.51	364736/0.54/0.62
0,1-partial rels.	935790/0.52/0.59	652752/0.64/0.72	802483/0.51/0.58	963530/0.49/0.56	812613/0.60/0.68
0,2-partial rels.	742778/0.60/0.66	336153/0.74/0.82	636660/0.58/0.65	593371/0.74/0.82	621128/0.74/0.82
1,0-partial rels.	889398/0.48/0.54	1095953/0.60/0.68	808649/0.47/0.53	1014837/0.44/0.50	865394/0.55/0.62
1,1-partial rels.	2049612/0.53/0.59	1817042/0.64/0.72	1817656/0.52/0.57	2189528/0.50/0.55	1930024/0.61/0.67
1,2-partial rels.	1628450/0.61/0.66	937005/0.74/0.81	1442146/0.59/0.65	1338927/0.75/0.81	1471358/0.75/0.82
2,0-partial rels.	623474/0.51/0.56	1071958/0.63/0.70	610361/0.50/0.55	622358/0.46/0.50	574197/0.57/0.63
2,1-partial rels.	1441725/0.56/0.61	1779998/0.68/0.74	1374875/0.55/0.60	1345966/0.51/0.55	1279510/0.63/0.68
2,2-partial rels.	1148798/0.64/0.68	916616/0.79/0.84	1094971/0.62/0.67	813549/0.78/0.82	972034/0.78/0.83
total relations	9868562/0.56/0.61	9001145/0.67/0.74	8947023/0.54/0.60	9328593/0.56/0.62	8890994/0.65/0.72
1.6li(L)	8348496	9219535	9219535	9219535	9219535
sieving time (days)	90	215	281	68	162

Table 2.8: Examples (continued)

In Table 2.6 we give the polynomials used for the sieving and the cont_p's for primes dividing the resultant of the polynomials.

In Tables 2.7 and 2.8, the triple entries a/b/c for the i,j-partial relations contain

$$\begin{aligned} a &:= \text{number } r_{i,j} \text{ of } i,j\text{-partial relations,} \\ b &:= \frac{X \cdot G_i(\alpha'_1, \beta'_1) \cdot G_j(\alpha'_2, \beta'_2)}{r_{i,j}} \\ c &:= \frac{X \cdot H_i(x'_1, B_1, L) \cdot H_j(x'_2, B_2, L)}{r_{i,j}} \end{aligned}$$

with $x_k = 2\overline{F_k}$ (see 2.4) and $x'_k = x_k \cdot e^{\alpha(F_k, B_k)}$, $\alpha'_k = \log_{x'_k} B_k$, $\beta'_k = \log_{x'_k} L$ for $k = 1, 2$. The time unit is a day.

We also state the value of $1.6\mathrm{li}(L)$ which is a heuristic estimate by Montgomery (private communication) of the number of total relations needed when sieving with large prime bound L. All the examples were tuned with simulations to yield approximately that number of total relations.

Examples 3,413+ and 3,427+ used the same higher-degree polynomial.

The polynomials for the two GNFS examples were chosen to have many factors modulo small primes. This is reflected by the negative α for the high degree polynomials.

The estimates with G_i vary from 44% to 79%. The H_i estimates are from 4% to 10% higher than the G_i estimates. The estimates tend to be lower for full relations than for partial relations with many large primes. The estimated number of total relations varies from 54% to 69% and 60% to 77% for G and H, respectively.

If one likes to know whether certain parameters yield enough data with the two-large-primes sieve without sieving, we suggest to tune the parameters to yield approximately $0.6 \cdot 1.6\mathrm{li}(L)$ or $0.7 \cdot 1.6\mathrm{li}(L)$ estimated total relations, respectively for G and H.

The sieving time seems to be hardly correlated with the numbers or parameters. This may be due to the use of different machines.

2.9 Obstructions when going from two to three large primes

In this section we use B and L without indices meaning the factor base bound and the large prime bound of the polynomial allowing three large primes. In the sequel, by candidate bi- or tri-composite we mean cofactors in $[B^2, L^2]$ and $[B^3, L^3]$, respectively. A candidate bi-composite cofactor can be either bi-composite or prime, a candidate tri-composite can be tri-composite, prime or bi-composite (see Figure 2.1 in Section 2.3).

We distinguish two types of bi-composites: either both primes are below the large prime bound L or one prime exceeds L. We discard the bi-composites of the second type. All bi-composite cofactors between B^3 and L^3 have at least one factor exceeding L (since we assumed $L^2 < B^3$), so we will discard those. Similarly among the tri-composites, we keep the ones with all three primes below L and discard the ones with at least one factor larger than L. The major obstruction when switching from two to three large primes is that only a small fraction of the composite candidate tri-composites really will be useful tri-composites, while most of them will be useless bi-composites.

Filtering out the primes is easy, as probable prime tests can be performed in times orders of magnitude smaller than what factoring takes. Unfortunately, distinguishing between bi- and tri-composites is not so quick on average. A known but not very fast method is to trial-divide primes starting from B to the cubic root of the number to be factored. The massive presence of bi-composites among the composite candidate tri-composites has a big impact on the average sieving time per useful relation, since a lot of effort is put into the factoring of cofactors which are not useful.

We are interested in a factorisation method which detects tri-composites quickly and gives up on factoring bi-composites in $[B^3, L^3]$ early. We found that Pollard's $P-1$ method is well-suited giving a good yield for numbers of the size of our candidate tri-composites.

The method finds factors p where $p-1$ has all factors below a given limit K_1 and possibly one factor between K_1 and a second limit K_2. For a description of the $P-1$ implementation, see Section 2.3.1. We are interested in small limits, as this means quitting the factorisation of bi-composites in $[B^3, L^3]$ early. On the other hand, we want large enough limits to guarantee that a high percentage of useful tri-composites will be found. Actually, only one factor needs to be found. If it is possible to find a fraction f of all the prime factors of the useful tri-composites, we estimate that a fraction $1-(1-f)^3$ of the useful tri-composites can be identified. We computed these fractions and the actual numbers of factored tri-composites for a series of 292 useful tri-composites in the interval $[10^{21}, 10^{27}]$ with factor base 10^7 and large prime bound 10^9 and different $P-1$ limits. Some results are reported in Table 2.9. We do not give the time for factoring the tri-composites here, as this is negligible compared with the time for factoring the bad bi-composites. Good limits can be investigated with a few simulation runs of the siever. The default values chosen for the implementation of $P-1$ are 2 000 and 50 000. For these values and for $B = 10^7$ and $L = 10^9$ only 50% of the factors of the useful tri-composites were found, but this accounts for the partial factoring of 87% of the useful tri-composites.

Table 2.10 compares some three-large-primes sieved runs for different $P-1$ bounds. We used the number $2^{773}+1$ (see Section 2.10). We sieved a sublattice of the sieving region used, namely the points $(a, 9973 \cdot b)$ with a an integer in $[-28\,875\,000, 28\,875\,000)$ and $b = 1, \ldots, 2200$. The first polynomial was allowed to have three large primes. The factor base bound was $B = B_1 = 2 \cdot 10^7$, the large prime bound $L = 10^9$.

A total of 1 765 748 candidate tri-composites were marked, 947 992 of which resulted prime. After checking that the cofactor from the other polynomial

K_1	K_2	#factors found	f	$1-(1-f)^3$	#tri-composites found	%
500	10000	247	0.28	0.63	183	0.63
500	20000	299	0.34	0.71	205	0.70
1000	20000	332	0.38	0.76	221	0.76
1000	50000	404	0.46	0.84	244	0.84
1200	60000	429	0.49	0.87	252	0.86
2000	50000	434	0.50	0.87	255	0.87
2000	100000	488	0.56	0.91	265	0.91
10000	100000	509	0.58	0.93	267	0.91
2000	200000	540	0.62	0.94	277	0.95
5000	250000	568	0.65	0.96	280	0.96
10000	500000	621	0.71	0.98	281	0.96
12000	600000	630	0.72	0.98	281	0.96
20000	1000000	676	0.77	0.99	285	0.98
50000	1000000	676	0.77	0.99	285	0.98
20000	1500000	702	0.80	0.99	288	0.99
50000	1500000	702	0.80	0.99	288	0.99
50000	2500000	730	0.83	1.00	290	0.99
60000	3000000	737	0.84	1.00	291	1.00

Table 2.9: Percentages found tri-composites for some $P-1$ limits

value is okay (a bi-composite candidate there will be factored before attempting the factorisation of the tri-composite), a total of 60 531 tri-composite candidates were tried to be factored by $P-1$. In the following listing we describe what the first 6 columns in Table 2.10 mean:

K_1 and K_2 These are the limits for the $P-1$ method.

not factored This gives the number of composites for which the $P-1$ method could not find a factor.

factor too large Here either the factor found by $P-1$ is too large or the remaining cofactor (prime or composite—not tested here) is too large. This count also includes tri-composites where the second found factor (by SQUFOF or Pollard Rho) or the corresponding cofactor is too large.

cofactor prime After finding the first factor, the size of the factor and the cofactor is checked (this is covered by the previous column). If okay, a probable prime test is performed on the cofactor. This column gives the number of probable prime cofactors. A prime cofactor corresponds to a bi-composite with a too large factor (because of $L^2 < B^3$).

three factors All three factors are smaller than L. These are the wanted tri-composites.

K_1	K_2	not factored	factor too large	cofactor prime	three factors	step 1 successful	step 2 successful	relations found	time per relation (s)
1000	50000	43158	6669	9814	890	3991	13382	3699	6.73
1200	60000	41862	7408	10345	916	4549	14120	3725	6.70
2000	100000	38213	9583	11770	965	6402	15916	3774	6.71
5000	250000	31721	13791	14014	1005	10478	18332	3814	6.90
10000	500000	27037	17058	15410	1026	14008	19486	3835	7.25
12000	600000	25877	17892	15730	1032	15022	19632	3841	7.41
20000	1000000	22780	19994	16715	1042	17903	19848	3851	7.95
50000	2500000	17746	23535	18198	1052	23182	19603	3861	9.87
60000	3000000	16878	24129	18471	1053	24212	19441	3862	10.62

Table 2.10: Tri-composite factorizations for $2^{773}+1$

The last two columns of Table 2.10 give the total number of relations found and the average time (in seconds) to find such a relation on a Silicon Graphics Origin 2000 MIPS R12000 300MHz. The lowest time on this list is with $P-1$ bounds 1 200 and 60 000. These are also the bounds used in the factorisation of $2^{773}+1$ (see Section 2.10).

We can see that for $P-1$ bounds 1 200 and 60 000, more than 10 times as many useless bi-composites than useful tri-composites were found. Note that the factor 10 is a rough lower bound for the ratio between bi- and tri-composites, as most of the numbers falling into column 3 and 4 are also bi-composites. The time per relation augments for larger $P-1$ bounds as more time-expensive bi-composites get factored and the number of tri-composites saturates. With $K_1 = 1\,200$ and $K_2 = 60\,000$, in average a $P-1$ run on a composite candidate tri-composite was about 61 times the time of a probable prime test on a candidate tri-composite.

In all the sieving experiments we reduced the intervals for bi-composites $[B_i^2, L_i^2]$, for $i = 1, 2$, and for tri-composites $[B_i^3, L_i^3]$, for i equal to 1 or 2, according to (2.2). This cuts down the sieving time as the search in the central parts is more effective. This can be seen in Table 2.11. The number sieved is $2^{773}+1$ on the same sublattice as above with $K_1 = 1\,200$ and $K_2 = 60\,000$. In this table, every line gives data considering only candidate tri-composites in $[B^vL^{3-v}, B^{v-0.5}L^{3.5-v}]$ while keeping the whole interval for the bi-composites, $[B^2, L^2]$. The interval $[B^{2.2}L^{0.8}, B^{1.1}L^{1.9}]$ from (2.1) is contained in $[B^{2.5}L^{0.5}, B^{1.0}L^{2.0}]$, which data is given by line 2 to line 4 of Table 2.11 and which have better rates. The time is for a SGI Origin 2000 MIPS R12000 300MHz processor.

v	tri-composite candidates	primes	not factored	factor too large	cofactor prime	three factors	step 1 successful	step 2 successful	relations found	time per relation (s)
3.0	499077	289805	10290	2344	3280	26	1305	4345	2851	8.5
2.5	635208	355779	14028	2619	3996	165	1645	5135	2990	8.1
2.0	788546	426673	18584	3194	4688	429	1972	6339	3254	7.5
1.5	876915	459383	21785	3922	4916	441	2290	6989	3266	7.5
1.0	628643	318788	15923	3927	2424	131	1592	4890	2956	8.4
0.5	57466	28589	1487	469	98	0	135	432	2825	8.6

Table 2.11: Tri-composite factorizations for cofactors in $[B_1^v L^{3-v}, B_1^{v-0.5} L^{3.5-v}]$

2.10 An example with three large primes

In this section, we test how well we can approximate the number of relations, especially the ones with three large primes. We consider a simplified case. Instead of sieving both polynomials simultaneously, we sieve them separately.

We use the special number $2,773+ = 2^{773}+1$ for this experiment. This 233-digit number was factored [62] in October, 2000 by the NFS using a linear and a degree-6 polynomial. The factor base bounds where $B_1 = B_2 = 2 \cdot 10^7$ and the large prime bound $L = 10^9$. On the linear side three large primes were allowed, on the other side two large primes. The sieving region (2.1) had $A = 28\,875\,000$ and $B = 22\,000\,000$. Further, we chose $S_1 = 0.1$ and $S_2 = 6.0$. The $P-1$ bounds (see Section 2.3.1) were set to 1 200 and 60 000.

We sieved values for the linear homogeneous polynomial $F_1(a,b) = a - 2^{129}b$ and the degree-6 polynomial $F_2(a,b) = a^6 + 2b^6$ separately.

The discriminant of $f_2(x) = x^6 + 2$ is divisible by 2 and 3, so we calculate $\mathrm{cont}_2(F_2)$ and $\mathrm{cont}_3(F_2)$ manually. Modulo 2 the polynomial is x^6 which has 0 as the only multiple root which means $n_2 = 1$. The polynomial has no roots modulo 2^2, so $n_{2^2} = n_{2^3} = \cdots = 0$. It follows that $\mathrm{cont}_2(F_2)$ consists only of the term (2.5) with $k=1$, so $\mathrm{cont}_2(F_2) = \frac{n_2}{2+1} = \frac{1}{3}$. In an analogous way we find that $\mathrm{cont}_3(F_2) = \frac{n_3}{3+1} = \frac{1}{2}$. The correction values are $\alpha(F_1, B_1) = 0.569915$ and $\alpha(F_2, B_2) = 1.938592$.

For this example, we sieved over a small part of the sieving region,[4] namely

[4]This is because we could not take the actual siever output as we were sieving the polynomials separately.

all integer pairs

$$(a, b) \in [-A, A) \times [1000001, 1000100] \quad \text{with} \quad \gcd(a, b) = 1. \tag{2.33}$$

This corresponds to about $X = 2A \sum_{1000001}^{1000100} \frac{\phi(b)}{b} \approx 3.52772 \cdot 10^9$ candidate pairs. The mean value of polynomial F_1 over this region is

$$\overline{F_1} = \frac{1}{2AB} \int_{-A}^{A-1} \int_{1000001}^{1000100} |F_1(a, b)| \, da \, db \approx 6.73793 \cdot 10^{44},$$

whereas $\overline{F_2} \approx 8.197267 \cdot 10^{43}$. We put $x_i = 2 \cdot \overline{F_i}$ for $i = 1, 2$. We assume (see Assumption 1) we have got to do with random numbers of maximal size $x_1' = x_1 e^{\alpha(F_1, B_1)} \approx 2.38 \cdot 10^{45}$ and $x_2' = x_2 e^{\alpha(F_2, B_2)} \approx 1.14 \cdot 10^{45}$.

Table 2.12 gives the results for the linear polynomial. The real siever did not find all good (a, b) pairs. Most of those missed are with three large primes, but also a few with two large primes were missed. Some were discarded because the unsieved part does not belong to one of the intervals given in (2.2). Others were missed because no factor can be found by the $P - 1$ method according to the chosen bounds (see 2.3.1). Another reason for missing a pair is that more small primes appear in the factorisation than anticipated with the choices of S_1 and S_2. Therefore we also ran a special (expensive) sieve which found all smooth numbers so that we can better compare with the theoretical expectations. The numbers of relations from the real siever are reported in column 3 of Table 2.12 whereas column 2 gives the results from the ideal siever. The estimates outnumber the number of relations from the ideal siever. The values of the fractions XG_i/R_i decrease when i increases. The same happens with the corresponding fraction with H_i instead of G_i.

			x_1		x_1'	
i	R_i	real	XG_i/R_i	XH_i/R_i	XG_i/R_i	XH_i/R_i
0	36214	36214	1.11	1.19	1.00	1.07
1	201002	201002	1.09	1.16	1.00	1.05
2	400217	397374	1.08	1.12	1.00	1.03
3	347230	184375	1.07	1.09	1.00	1.02
4	122983	0	1.04	1.05	0.99	1.00
5	11820	0	1.00	n.a.	1.00	n.a.

Table 2.12: Numbers of smooth values of the linear polynomial $F_1(a, b)$ with (a, b) satisfying (2.33)

Table 2.13 gives the data for the degree-6 polynomial. Only the ideal sieve data is given, but no substantial difference with the real data should be expected here as this polynomial was only sieved with two large primes and only a small part of the pairs with two large primes get discarded. Here, as in the examples from Section 2.7, we can see again that the approximations get better when more large primes are allowed.

		x_2		x'_2	
i	R_i	XG_i/R_i	XH_i/R_i	XG_i/R_i	XH_i/R_i
0	120758	0.48	0.52	0.34	0.37
1	577746	0.54	0.57	0.39	0.41
2	959430	0.61	0.64	0.46	0.48
3	663086	0.71	0.73	0.57	0.58
4	170016	0.88	0.88	0.76	0.77
5	10232	n.a.	n.a.	1.15	n.a.

Table 2.13: Numbers of smooth values of the degree-6 polynomial $F_2(a, b)$ with (a, b) satisfying (2.33)

The approximations are within 20% for the linear polynomial, but rather poor for the higher-degree polynomial (up to 66% off). This is because the linear polynomial is near-constant over all of (2.33) while the degree-six polynomial grows from $2 \cdot 10^{36}$ to $5 \cdot 10^{44}$. To get better results for the higher-degree polynomial we should split up the sieving region in smaller pieces and do the approximations on the smaller pieces. This is left for further research.

We gave the results from the approximations by using the real size of the numbers (x) as well as the size when comparing to random numbers (x'). For the linear case, the latter gives better results, for the other polynomial it is exactly the other way round. However, in both cases the estimates with x' are lower than the ones with x, due to the positive α.

2.10.1 Approximation for the number of smooth numbers in an interval

We investigate how our formulae work for numbers in an interval instead of for polynomial values. Let us choose the interval

$$[-X/2 + 2^{129} \cdot 1\,000\,100, X/2 + 2^{129} \cdot 1\,000\,100), \tag{2.34}$$

which treats the same number X of candidates as in the previous section. The numbers in this interval are larger than $x = 2\overline{F_1}$ but still of the same order of magnitude.

The approximations $xG_i(\log_x B_1, \log_x L)$ and $xH_i(x, B_1, L)$ actually approximate the portion of smooth numbers between 1 and x, so, for the special case of intervals, we will define G_i^{int} and H_i^{int} and use $XG_i^{\text{int}}(\log_x B_1, \log_x L)$ and $XH_i^{\text{int}}(x, B_1, L)$ with x being some element in the interval.

Let us construct G_i^{int} and H_i^{int}. As an estimate for smooth values in an interval $[x_l, x_r]$ we approximate $\Psi_i(x_r, y, z) - \Psi_i(x_l, y, z)$ by the derivative of the approximation (2.9)

$$G_i^{\text{int}}(x,z,y) = \frac{d}{dx}\left(xG_i(\log_x z, \log_x y)\right) =$$

$$G_i(\log_x z, \log_x y) - \frac{1}{i!}\int_z^y \cdots \int_z^y \frac{\rho\left(\frac{\log x - \log(t_1\cdots t_i)}{\log z} - 1\right)}{\log x - \log(t_1 \cdots t_i)} \frac{dt_1}{t_1 \log t_1} \cdots \frac{dt_i}{t_i \log t_i} \tag{2.35}$$

times $x_r - x_l$ [5] for $\log x \geq \log z + i \log y$, or the derivative of the approximation (2.10)

$$H_i^{\text{int}}(x,z,y) = \frac{d}{dx}\left(xH_i(x,z,y)\right) = H_i(x,z,y) -$$

$$\frac{1}{i!}\int_z^y \cdots \int_z^y \frac{\rho\left(\frac{\log x - \log(t_1\cdots t_i)}{\log z} - 1\right)}{\log x - \log(t_1 \cdots t_i)} \frac{dt_1}{t_1 \log t_1} \cdots \frac{dt_i}{t_i \log t_i} -$$

$$\frac{1-\gamma}{\log x}\frac{1}{i!}\int_z^y \cdots \int_z^y \left(\frac{\rho\left(\frac{\log x - \log(t_1\cdots t_i)}{\log z} - 1\right)}{\log x} + \frac{\rho\left(\frac{\log x - \log(t_1\cdots t_i)}{\log z} - 2\right)}{\log x - \log(t_1 \cdots t_i) - \log z}\right)$$

$$\frac{dt_1}{t_1 \log t_1} \cdots \frac{dt_i}{t_i \log t_i} \tag{2.36}$$

times $x_r - x_l$ for $\log x \geq 2\log z + i \log y$.

i	R_i	XG_i/R_i	XH_i/R_i	XG_i^{int}/R_i	XH_i^{int}/R_i
0	40920	1.11	1.19	0.91	0.98
1	223495	1.10	1.16	0.92	0.97
2	439114	1.09	1.13	0.93	0.97
3	374335	1.07	1.10	0.95	0.97
4	128293	1.05	1.05	1.01	n.a.

Table 2.14: Numbers of smooth numbers in interval (2.34) with $x = 2^{129} \cdot 1\,000\,000$, $z = B_1$ and $y = L$

Again, the estimates are within 20% from the real data. This is comparable with the results for linear polynomials (Table 2.12).

2.11 Comparing the two- and the three-large-primes method

The numbers $7,211- = 7^{211} - 1$ and $7,211+ = 7^{211} + 1$ differ by 2 and are therefore suited for comparison purposes. We sieved $7,211-$ while allowing two

[5] In our examples the interval bounds $x_r - x_l \ll x_l$, so it does not matter which $x \in [x_l, x_r]$ we use.

large primes on both polynomials whereas for $7,211+$ we allowed up to three large primes on the linear side, i.e. polynomial 1.

We did not take advantage of already known factors

$$\begin{aligned} 7^{211}-1 &= 2\cdot 3\cdot 141793\cdot c173 \\ 7^{211}+1 &= 2^3\cdot 255571219\cdot \\ &\quad 98601841563833114489770514915288716311083\,9\cdot c128 \end{aligned}$$

by taking for $7,211\pm$ the polynomials $f_1(x)=7^{42}x-1$ and $f_2(x)=x^5\pm 7$ with root 7^{-42} modulo $c128$ and $c173$, respectively.[6] So both numbers have the same SNFS difficulty. For both numbers we have $\mathrm{cont}_5(F_2)=\frac{3}{8}$ and $\mathrm{cont}_7(F_2)=\frac{1}{8}$. Both numbers had the same sieving region and used identical large prime bounds, but $7,211-$ had a large factor base with the linear side while $7,211+$ allowed three primes there.

There is a minor secondary effect of the known factors. Since, for example, $7^{211}+1$ is divisible by 8, the polynomial $f_1=7^{42}x-1$ and $f_2(x)=x^5+7$ share a root $x\equiv 1 \bmod 8$, increasing the likelihood that both are simultaneously smooth.

Sieving simulations indicated that sieving with three large primes would be more costly in time, see the value for the estimated time per relation in Table 2.15. For the $P-1$ method we used the default bounds $K_1=2\,000$ and $K_2=50\,000$ 2.3.1.

In Table 2.15, a 'i,j-partial rels.' entry gives the number of sieved i,j-partial relations in the first column as well as the estimates $X\cdot G_i(\log_{x_1'}B_1,\log_{x_1'}L)\cdot G_j(\log_{x_2'}B_2,\log_{x_2'}L)$ and $X\cdot H_i(x_1',B_1,L)\cdot H_j(x_2',B_2,L)$ in the second and third column, respectively. The values x_k' and $\alpha(F_k,B_k)$, $k=1,2$, are defined in Section 2.4.1. In the total relations entry we give the percentage of the real relations instead of the estimates themselves.

The detailed real sieving data for $7,211-$ have unfortunately been lost but we expect their ratios to the estimates to be comparable with those for 7,211+. The theoretical estimates for relations with fewer than three large primes for $7,211+$ vary between 61% and 86% of the real number of relations.

For the three-large-primes relations the estimates outnumber the real numbers, since (because of time considerations) the siever discards many three-large-primes candidates.

The real number of relations for $7,211+$ is smaller than expected from the sieving simulations. A 2% deviation could be expected, but in fact it is 5%.

We cannot use the total sieving time as an indicator for performance, since $7,211-$ was sieved exclusively on low-memory machines, which tend to be slower.

We wanted to analyse which set of relations would give the better (smaller and lighter) matrix when considering the same number of relations. To this end, we truncated each data set to 11.4 million non-duplicate relations. After that

[6]We inverted the root as it is more more convenient to have $A>B$ with line sieving.

method (# large primes)	2+2	3+2
name	7, 211−	7,211+
SNFS difficulty	179	179
cofactor size	173	128
degree $f_1(x)$	1	1
degree $f_2(x)$	5	5
A	5400000	5400000
B	3500000	3500000
X	$2.29796 \cdot 10^{13}$	$2.29796 \cdot 10^{13}$
B_1	15000000	6000000
B_2	6000000	6000000
L	120000000	120000000
S_1	300	1
S_2	100	100
x_1	$1.68466 \cdot 10^{42}$	$1.68466 \cdot 10^{42}$
x_2	$2.36785 \cdot 10^{33}$	$2.36785 \cdot 10^{33}$
$\alpha(F_1, B_1)$	0.569915	0.569915
$\alpha(F_2, B_2)$	1.027386	1.027386
full relations	-/210279/243333	119560/73402/85456
0,1-partial rels.	-/646312/731630	342560/225609/256941
0,2-partial rels.	-/654225/725637	295549/228372/254836
1,0-partial rels.	-/524958/596629	525659/326026/372459
1,1-partial rels.	-/1613508/1793884	1506575/1002072/1119873
1,2-partial rels.	-/1633264/1779189	1303147/1014341/1110699
2,0-partial rels.	-/480504/536882	831856/530064/594762
2,1-partial rels.	-/1476872/1614244	2394438/1629201/1788273
2,2-partial rels.	-/1494954/1601021	2077068/1649149/1773624
3,0-partial rels.	n.a.	318335/385258/425221
3,1-partial rels.	n.a.	917502/1184126/1278512
3,2-partial rels.	n.a.	795323/1198624/1268039
total relations	12112998/0.72/0.79	11427572/0.83/0.90
$1.6\text{li}(L)$	10947914	10947914
sieving time (days)	776	707
estim. time per rel. (simulation)	0.95s	1.04s
estim. # rels. (simulation)	12.0M–12.5M	11.8M–12.2M
# rels. with more th. 2 lin. pr. >6M	2 601 059	2 026 232
#ideals of norm >1M	11 137 981	11 096 890
matrix size	1 163 421 × 1 252 099	1 135 638 × 1 224 487
# non-dupl. rels. in matrix	3 529 432	3 509 616
# non-dupl. rels. in matrix with more th. 2 lin. pr. >6M	539 731	359 597
matrix weight	23 346 515	22 599 998

Table 2.15: Comparison of 7, 211− with 7, 211+

we filtered (see Chapter 3) with `mergelevel` 1, `filtmin` 1M and `keep` 200K and later with `mergelevel` 8, `filtmin` 500K, `maxrels` 13.0 and `maxdiscard` 40K.

The matrix of $7, 211+$ is lighter and slightly smaller than the matrix of $7, 211-$ as there are fewer relations with more than 2 linear primes larger than 6 million (the B_1 for $7, 211+$) in the 7,211+ matrix. If all the relations had been considered for the filtering this would probably have led to a smaller matrix for $7, 211-$.

We conclude from this experiment that the three-large-primes method was still not necessary, so the number $7, 211+$ could easily have been sieved by the two-large-primes method. In a further comparison experiment one might try equal factor base bounds for both the 2-large-primes and the 3-large-prime while having a smaller sieving region for the 3-large-primes bound.

2.12 Conclusions

The examples given in Section 2.8 show that we can reasonably well estimate the number of partial relations with the formulae provided and can use this for calculating how many total relations to expect in the two-large-primes method for given parameter choices. However, calculating the heuristic $\alpha(F_k, B_k)$ might be too cumbersome and so a short sieving experiment will usually be preferred. Moreover, a sieving simulation will also provide a global time estimate. To improve the estimates one would probably need to split the sieving region into smaller regions and calculate the mean absolute value of the polynomials over the smaller regions.

In Section 2.9 we describe the obstructions which are encountered when going from two to three large primes. These obstructions forced us to avoid the "ideal" three-large-primes method which would generate all possible three-large-primes relations and would consequently be too costly in time. Instead we chose for an approach which abandons unpromising candidates quickly.

Our theoretical estimates for the three-large-primes relations indicate how many relations would be obtained with the "ideal" siever and so give a useful measure of how far the real siever (with its parameter choices for $P-1$) is off from the "ideal" siever.

For the sieving of the record SNFS number $2, 773+$ (see Section 2.10) the three-large-primes method was convenient to keep the factor base small and equal for all participating sieving computers. However, it would also have been possible to sieve with the two-large-primes method with a larger factor base on machines with sufficiently large memory in combination with the three-large-primes method on small memory machines.

The comparison between the sieving of $7, 211-$ and $7, 211+$ which were sieved with the two-large-primes method and the three-large-primes method with a smaller factor base bound, respectively, did not show a significant difference between the two approaches.

The general number RSA-155 (see Appendix B) was still sieved with the two-large-primes method, though for a considerable part with the lattice siever with two large primes. That method can be seen as a kind of three-large-primes method because of the additional special prime.

The other 155-digit GNFS factorisation [2] was done with the line-by-line siever, presumably with 2 large primes. The sieving took longer than for RSA-155 but, apart from the choice of the siever, this may also be due to the polynomial and other parameter choices.

For further research it might be interesting to study the influence of the three-large-primes method on the matrix by sieving a number twice (or two similar numbers, as we did with $7,211-$ and $7,211+$), once with the two-large-primes method, once with the three-large-primes method while using identical parameters (in particular, also the factor base bound and the large primes bound are identical) except for the sieving region which can be smaller for the three-large-primes method.

Appendix 2.A. Proofs of formulae from Section 2.7

The following two propositions give the proofs for the two improved formulae for the Taylor coefficients of ρ.

Proposition 1 (Patterson Rumsey $c_i^{(k)}$). *Let $\rho(k-\xi) = \sum_{i=0}^{\infty} c_i^{(k)} \xi^i$ with $\xi \in [0,1]$. We have*

$$c_i^{(k)} = \frac{c_{i-1}^{(k-1)} + (i-1)c_{i-1}^{(k)}}{ki} \quad \textit{for} \quad i > 0 \quad \textit{and} \quad k > 1. \tag{2.37}$$

Proof. From (2.7) we know

$$\frac{d}{d\xi}(\rho(k-\xi)) = \frac{\rho(k-1-\xi)}{k-\xi}.$$

We substitute the Taylor series of ρ for the intervals $[k-1,k]$ and $[k-2,k-1]$, multiply by $k-\xi$ on both sides and get

$$(k-\xi)\frac{d}{d\xi}\left(\sum_{i=0}^{\infty} c_i^{(k)}\xi^i\right) = \sum_{i=0}^{\infty} c_i^{(k-1)}\xi^i.$$

If the sums are uniformly convergent, we can differentiate term by term which leads to

$$k\sum_{i=0}^{\infty} ic_i^{(k)}\xi^{i-1} - \sum_{i=0}^{\infty} ic_i^{(k)}\xi^i = \sum_{i=0}^{\infty} c_i^{(k-1)}\xi^i.$$

On comparing coefficients we obtain

$$c_i^{(k)} = \frac{c_{i-1}^{(k-1)} + (i-1)c_{i-1}^{(k)}}{ki}.$$

□

Proposition 2 (Marsaglia Zaman Marsaglia $c_0^{(k)}$). *Let* $\rho\left(k+\frac{1}{2}+\frac{1}{2}\xi\right)=\sum_{i=0}^{\infty} c_i^{(k)}\xi^i$ *with* $\xi\in[-1,1]$. *We have*

$$c_0^{(k)}=\frac{1}{2k}\left(c_0^{(k-1)}+\sum_{j=1}^{\infty}\frac{c_j^{(k-1)}+(-1)^j c_j^{(k)}}{j+1}\right) \quad \textit{for} \quad k>0. \tag{2.38}$$

Proof. Because of

$$\rho(x)=\frac{1}{x}\int_{x-1}^{x}\rho(t)dt \quad \text{for} \quad x>1 \tag{2.39}$$

(which is another way of defining Dickman's ρ function for $x>1$) we can write

$$c_0^{(k)}=\rho\left(k+\frac{1}{2}\right)=\frac{1}{k+\frac{1}{2}}\int_{k-\frac{1}{2}}^{k+\frac{1}{2}}\rho(t)dt.$$

We split the integral in an integral from $k-\frac{1}{2}$ to k and one from k to $k+\frac{1}{2}$. Next we substitute $t=k-\frac{1}{2}+\frac{1}{2}z$ and $t=k+\frac{1}{2}+\frac{1}{2}z$, respectively. Hence,

$$c_0^{(k)}=\frac{1}{2k+1}\left(\int_0^1\rho\left(k-\frac{1}{2}+\frac{1}{2}z\right)dz+\int_{-1}^0\rho\left(k+\frac{1}{2}+\frac{1}{2}z\right)dz\right).$$

We substitute the respective Taylor expansions and integrate to obtain

$$c_0^{(k)}=\frac{1}{2k+1}\left(\sum_{j=0}^{\infty}\frac{1}{j+1}\left(c_j^{(k-1)}+c_j^{(k)}(-1)^j\right)\right)$$

which implies

$$c_0^{(k)}=\frac{1}{2k}\left(c_0^{(k-1)}+\sum_{j=1}^{\infty}\frac{c_j^{(k-1)}+(-1)^j c_j^{(k)}}{j+1}\right).$$

□

For completeness we also give the proofs for the remaining two recurrence relations. The proof of Proposition 4 is taken from [4].

Proposition 3 (Marsaglia Zaman Marsaglia $c_i^{(k)}$). *Let* $\rho\left(k+\frac{1}{2}+\frac{1}{2}\xi\right)=\sum_{i=0}^{\infty} c_i^{(k)}\xi^i$ *with* $\xi\in[-1,1]$. *We have*

$$c_i^{(k)}=-\frac{c_{i-1}^{(k-1)}+(i-1)c_{i-1}^{(k)}}{i(2k+1)} \quad \textit{for} \quad i>0 \quad \textit{and} \quad k>0. \tag{2.40}$$

Proof. From (2.7) we know

$$\frac{d}{d\xi}\left(\rho\left(k+\frac{1}{2}+\frac{1}{2}\xi\right)\right) = -\frac{1}{2}\frac{\rho\left(k-\frac{1}{2}+\frac{1}{2}\xi\right)}{k+\frac{1}{2}+\frac{1}{2}\xi}.$$

We substitute the Taylor series of ρ for the intervals $[k, k+1]$ and $[k-1, k]$, multiply by $2k+1+\xi$ on both sides and get

$$(2k+1+\xi)\frac{d}{d\xi}\left(\sum_{i=0}^{\infty} c_i^{(k)}\xi^i\right) = -\sum_{i=0}^{\infty} c_i^{(k-1)}\xi^i.$$

If the sums are uniformly convergent, we can differentiate term by term which leads to

$$(2k+1)\sum_{i=0}^{\infty} i c_i^{(k)}\xi^{i-1} + \sum_{i=0}^{\infty} i c_i^{(k)}\xi^i = -\sum_{i=0}^{\infty} c_i^{(k-1)}\xi^i.$$

On comparing coefficients we obtain

$$c_i^{(k)} = -\frac{c_{i-1}^{(k-1)} + (i-1)c_{i-1}^{(k)}}{i(2k+1)}.$$

□

Proposition 4 (Patterson Rumsey $c_0^{(k)}$). *Let $\rho(k-\xi) = \sum_{i=0}^{\infty} c_i^{(k)}\xi^i$ with $\xi \in [0,1]$. We have*

$$c_0^{(k)} = \frac{1}{k-1}\sum_{j=1}^{\infty}\frac{c_j^{(k)}}{j+1} \quad \textit{for} \quad k > 1. \tag{2.41}$$

Proof. Because of (2.39) we have

$$c_0^{(k)} = \rho(k) = \frac{1}{k}\int_{k-1}^{k}\rho(t)dt = \frac{1}{k}\int_0^1 \rho(k-z)\,dz.$$

We substitute the Taylor expansion and integrate to obtain

$$c_0^{(k)} = \frac{1}{k}\left(\sum_{j=0}^{\infty}\frac{1}{j+1}c_j^{(k)}\right)$$

which implies

$$c_0^{(k)} = \frac{1}{k-1}\sum_{j=1}^{\infty}\frac{c_j^{(k)}}{j+1}.$$

□

In all four propositions we implicitly assumed that sums are uniformly convergent. In fact, it can be proven by induction that the radius of convergence equals 2 for the series defined inductively by (2.37) and (2.41) with start series $c_0^{(1)} = 1$ and $c_i^{(1)} = 0$ for $i > 0$, and 3 for the series defined inductively by (2.40) and (2.38) and start series $c_0^{(0)} = 1$ and $c_i^{(0)} = 0$ for $i > 0$.

Chapter 3

Strategies in Filtering in the Number Field Sieve

Abstract

A critical step when factoring large integers by the Number Field Sieve [28] consists of finding dependencies in a huge sparse matrix over the field $\mathbb{F}_2$, using a Block Lanczos algorithm. Both size and weight (the number of non-zero elements) of the matrix critically affect the running time of Block Lanczos. In order to keep size and weight small the relations coming out of the siever do not flow directly into the matrix, but are filtered first in order to reduce the matrix size. This paper discusses several possible filter strategies and their use in the recent record factorisations of RSA-140, R211 and RSA-155.

3.1 Introduction

The Number Field Sieve (NFS) is the asymptotically fastest algorithm known for factoring large integers. It holds the records in factoring special numbers (R211 [61]) as well as general numbers (RSA-140 (see Appendix A) and RSA-155 (see Appendix B). One disadvantage is that it produces considerably larger matrices than other methods, such as the Quadratic Sieve [8]. Therefore it is more and more important to find ways to limit the matrix size. This can be achieved by using good sieving parameters and by "intelligent" filtering.

In this paper we describe the extended version of the program `filter` which we implemented following ideas of Peter L. Montgomery. Its goal is to speed up Block Lanczos's running time by reducing the matrix size but still keeping the weight under control.

This chapter is a slight revision of an article which appeared in the proceedings of the ANTS IV conference [15].

A previous implementation of the program `filter` [28, section 7] did 2- and 3-way merges. When using Block Lanczos, higher-way merges were commonly banned from the filter step in order to limit the matrix weight. For instance, also James Cowie et al. [20, section Cycles] explicitly avoided merges higher than 3 for the factorisation of RSA-130.

The most important new ingredients of the present `filter` implementation are an algorithm to discard excess relations and "controlled" higher-way merges. We determine arithmetically which merges reduce Block Lanczos's running time.

For the factorisation of RSA-140 only 2- and 3-way merges were performed which led to a matrix of 4.7 million columns. With the present filter strategy we could have saved up to 33% of linear algebra time by reducing the size to 3.3 million columns. For the factorisation of R211 we already used an intermediate `filter` version which did 4- and 5-way merges, but we could still get an improved matrix after the factorisation. For RSA-155, we could take full advantage of the present version and did "controlled" merges up to prime ideal frequency 8 which led to a matrix of 6.7 million columns and an average of 62 entries per column which was used to factor the number. Afterwards, we were able to reduce this size to 6.3 million columns.

First, we give a brief description of the NFS. Secondly, the `filter` implementation will be described with special focus on the new features. In section 3.4 we will describe other filter strategies we came across in the literature and compare it with our approach. Finally, experimental results for RSA-140, R211 and RSA-155 are listed and interpreted.

3.2 Brief description of NFS

We briefly describe the NFS factoring method here, skipping parts which are not relevant for the understanding of this paper such as the sieving step itself.

By N we denote the composite number we would like to factor. We select an integer M and two irreducible polynomials $f(x)$ and $g(x) \in \mathbb{Z}[x]$ with $\mathrm{cont}(f) = \mathrm{cont}(g) = 1$ and $f \neq \pm g$ such that $f(M) \equiv g(M) \equiv 0 \bmod N$. By $\alpha, \beta \in \mathbb{C}$ we denote roots of $f(x)$ and $g(x)$, respectively.

The goal is to construct a non-empty set S of co-prime integer pairs (a, b) for which both $\prod_{(a,b)\in S}(a - b\alpha)$ and $\prod_{(a,b)\in S}(a - b\beta)$ are squares, say, $\gamma^2 \in \mathbb{Z}[\alpha]$ and $\delta^2 \in \mathbb{Z}[\beta]$, respectively. Once we have found S, the two natural ring homomorphisms $\phi_1 : \mathbb{Z}[\alpha] \to \mathbb{Z}/N\mathbb{Z}$ mapping α to M and $\phi_2 : \mathbb{Z}[\beta] \to \mathbb{Z}/N\mathbb{Z}$ mapping β to M as well, yield the congruence

$$\phi_1(\gamma)^2 \equiv \phi_1(\gamma^2) \equiv \prod_{(a,b)\in S} (a - bM) \equiv \phi_2(\delta^2) \equiv \phi_2(\delta)^2 \bmod N.$$

which has the desired form $X^2 \equiv Y^2 \bmod N$. By computing $\gcd(X - Y, N)$ we may find a divisor of N. The major obstruction in this series of congruences is that we need to find $\gamma \in \mathbb{Q}(\alpha)$ from γ^2 (and δ from δ^2, respectively). See

Montgomery's [48] or Phong Nguyen's [55] papers for a description of their square root algorithms.

How to find the set S? We write

$$F(x, y) = f(x/y)y^{\deg(f)} \quad \text{and} \quad G(x, y) = g(x/y)y^{\deg(g)}$$

for the homogeneous form of $f(x)$ and $g(x)$, respectively. Consider $a - b\alpha \in \mathbb{Q}(\alpha)$ and $a - b\beta \in \mathbb{Q}(\beta)$. The minus sign is chosen in order to have

$$N_{\mathbb{Q}(\alpha)/\mathbb{Q}}(a - b\alpha) = F(a, b)/c_1 \quad \text{and} \quad N_{\mathbb{Q}(\beta)/\mathbb{Q}}(a - b\beta) = G(a, b)/c_2,$$

where the c_i's are the respective leading coefficients of $f(x)$ and $g(x)$.

After the sieving we are left with many pairs (a, b) such that $\gcd(a, b) = 1$ and both $F(a, b)$ and $G(a, b)$ are products of primes smaller than the large prime bounds L_1 and L_2, respectively, which were chosen by the user before the sieving. The pairs (a, b) are commonly denoted as *relations*. A necessary condition for

$$\prod_{(a,b)\in S} (a - b\alpha) \quad \text{and} \quad \prod_{(a,b)\in S} (a - b\beta)$$

to be squares is that the norms

$$N_{\mathbb{Q}(\alpha)/\mathbb{Q}}\left(\prod_{(a,b)\in S} (a - b\alpha)\right) \quad \text{and} \quad N_{\mathbb{Q}(\beta)/\mathbb{Q}}\left(\prod_{(a,b)\in S} (a - b\beta)\right)$$

are squares. Therefore we require S to have even cardinality and

$$\prod_{(a,b)\in S} F(a, b) \quad \text{and} \quad \prod_{(a,b)\in S} G(a, b)$$

to be squares. The condition is not sufficient because elements having the same norm may differ from each other (not only by units!). Let p be a prime divisor of $F(a, b) = f(a/b)b^{\deg(f)}$. We distinguish two cases:

- $p \mid f(a/b)$. This means that $a/b \equiv q \bmod p$ with $0 \le q < p$ is a root of $f(x)$ modulo p. In the sequel such a p is referred to as p, q.
- $p \mid b$. Since $\gcd(a, b) = 1$ it follows that $p \nmid a$ and therefore $p \mid c_1$. This can happen for a small set of primes only, since the leading coefficient is of limited size. These roots are called *projective roots* and denoted as p, ∞.

We will call the couples p, q, where q is allowed to be ∞, *prime ideals*, since they are in bijective correspondence with the first degree prime ideals of the ring $\mathbb{Z}[\alpha] \cap \mathbb{Z}[\alpha^{-1}]$. See [13, Section 12.6].

Consequently, we write

$$|F(a, b)| = \prod_{p,q} p^{e_1(a,b,p,q)} \quad \text{and} \quad |G(a, b)| = \prod_{p,q} p^{e_2(a,b,p,q)}.$$

In order for $\prod_{(a,b)\in S} F(a,b)$ and $\prod_{(a,b)\in S} G(a,b)$ to be squares in $\mathbb{Q}(\alpha)$ and $\mathbb{Q}(\beta)$, respectively, we require all the exponents in

$$\prod_{(a,b)\in S} |F(a,b)| = \prod_{p,q} p^{\sum_S e_1(a,b,p,q)} \quad \text{and} \quad \prod_{(a,b)\in S} |G(a,b)| = \prod_{p,q} p^{\sum_S e_2(a,b,p,q)}$$

to be even. This condition can be stated in terms of the field $\mathbb{F}_2$ as well. We just think of a relation (a,b) as a vector in $\mathbb{F}_2$ whose first entry is 1 (in order to control the parity of S) and the following entries are given by the exponents $e_1(a,b,p,r)$ and $e_2(a,b,p,r)$ modulo 2. A 1 signals the occurrence of an uneven power of a prime ideal. The task of finding some suitable sets S translates now into finding dependencies modulo 2 between the columns of a matrix which is built up with the relation vectors given by the siever. We need to have enough relations to guarantee that the matrix provides enough dependencies.

Alas, not every dependency yields a set S such that $\prod_{(a,b)\in S}(a - b\alpha)$ and $\prod_{(a,b)\in S}(a - b\beta)$ are squares, but we can make the method practical by producing several dependencies and doing quadratic character tests [13, Section 8].

The filter stage occurs between the sieving step and the linear algebra step of the NFS. It is a preliminary linear algebra process since it corresponds to dropping columns (*pruning*) and adding up columns modulo 2 (*merging*).

3.3 Description of the new filter tasks

We distinguish 19 merge levels: level 0 and 1 fall into pruning, level 2 through 18 within merging.

We shall say that a prime ideal p, q is *(un)balanced* in a relation (a,b) if it appears to an (un)even number in $F(a,b)$ or $G(a,b)$.[2] We distinguish between prime ideals of norm below and above a user determined bound `filtmin`. Accordingly, we speak about *small* and *large* prime ideals. We will denote prime ideals p, q by I. We write a relation $r = r(a,b)$ as the collection of its unbalanced large prime ideals, $r : I_1, I_2, \ldots, I_k$. Merging means combining relations which have a common prime ideal in order to balance it. For example, if I appears only in $r_1 : I_{10} = I, I_{11}, \ldots, I_{1k_1}$ and $r_2 : I_{20} = I, I_{21}, \ldots, I_{2k_2}$, we can combine the two relations into $r_1 + r_2 : I_{11}, \ldots, I_{1k_1}, I_{21}, \ldots, I_{2k_2}$ with the result that I is balanced in $r_1 + r_2$. More generally, a *k-way merge* is the procedure of combining k relations with a common prime ideal I into $k-1$ relation pairs without I. By a *relation-set* we mean a single relation, or a collection of two or more relations generated by a merge. We do merges up to prime ideal frequency 18. The parameter `mergelevel` l means that k-way merges with $k \leq l$ may be performed. The weight of a relation-set r, i.e., the number of unbalanced prime ideals in it, is denoted by $w(r)$.

[2] In very rare cases (p divides the polynomial resultant) we can have the same p, q appearing in both F and G. Recall that they are *not* the same, since they correspond to ideals in different rings. We abstain from labelling the ideals accordingly, for the sake of simplicity.

3.3.1 Pruning

As the verb "pruning" suggests, this part of the program removes unnecessary relations from the given data, that is *duplicates* and *singletons* and, if the user wants to, also excess relations. Duplicates are obviously superfluous and singletons cannot be part of a winning set S since they contain a prime ideal which does not occur in any other relation and can subsequently not be combined to form a square. If the difference between the number of relations and the number of large prime ideals outnumbers a user-chosen bound (`keep`), the *clique* algorithm selects relations to delete.

`mergelevel` 0 only removes duplicates and can be used to merge several sieving outputs to a single file, possibly before sieving completes. `mergelevel` 1 will only be performed if the full set of relations is available and covers algorithms for the removal of duplicates, singletons and excess relations.

Duplicates.

First we want to eliminate duplicate relations. They may arise for various reasons. Most commonly they come from sieving jobs that were stopped and later restarted. In case of a *line-by-line siever* [28, section 6] the resumed jobs start with the last b sieved by the previous job; this is the only way that duplicates arise. In case of a *lattice siever* [57] the job starts with the special prime ideal I sieved last, and will generate duplicates, or it can do so because a relation may contain, apart from its own special I, other prime ideals that are used as special prime ideals as well. The simultaneous use of line-by-line and lattice siever also causes overlap.

Duplicates are tracked down by hashing [35]. Since it is easier and cheaper to use a number instead of a relation as a hash table entry, we "identify" a relation with a number. The user specifies how many relations he expects to be in the input file(s) (`maxrelsinp`). This figure is used to choose the size of the in-memory tables needed during the pruning algorithm. The program reads in relation after relation. In order to detect duplicates, the program maps each relation (a, b) to an integer between 0 and $2^{64} - 1$. The mapping function, $h = h(a, b)$, should be nearly injective since relations mapped to the same value will be treated as duplicates. It is rather easy to construct such a function, since even a huge amount of relations, say 200 million (for RSA-155 we had to handle 124.7 million relations), is small compared to the 2^{64} possible function values. With 64 bits for the function value we expect about

$$\frac{\binom{2\cdot 10^8}{2}}{2^{64}} \approx 0.0011$$

false duplicates, which means that there will hardly be any false duplicates. With 32 bits only, this number would amount to about $4.7 \cdot 10^6$, which is a fair proportion of all relations.

The function $h(a,b)$ is defined as follows. It takes values of a and b up to 2^{53}. Put $\Pi = \lfloor \pi \cdot 10^{17} \rfloor$ and $E = \lfloor e \cdot 10^{17} \rfloor$. We have $\gcd(\Pi, E) = 1$. Define

$$H(a,b) = \Pi a + Eb.$$

If $H(a_1,b_1) = H(a_2,b_2)$ and $(a_1,b_1) \neq (a_2,b_2)$ we have

$$\frac{a_1 - a_2}{b_1 - b_2} = -\frac{E}{\Pi}$$

which is impossible, since $|a|$ and $|b|$ are known to be much smaller than $\Pi/2$ and $E/2$, and $\gcd(\Pi, E) = 1$. Define $h(a,b) = H(a,b) \bmod 2^{64}$. Since H is injective, false duplicates for h can only come from the truncation modulo 2^{64}.

The function values of h again are mapped by a hash function into a hash table. If the user has specified `mergelevel` 0, the non-duplicates are written to the output file whereas, if the user has chosen `mergelevel` 1, the non-duplicate relations are memorised in a table for further processing, while considering only the large prime ideals. In the sequel, we shall call this table the *relation table.*

Singletons.

If both polynomials f and g split completely into distinct linear factors modulo a prime p which does not divide the leading coefficients, we get a so-called free relation corresponding to the prime ideal factorisation of the elements $p = p - 0\alpha$ and $p = p - 0\beta$ of norm $N_{\mathbb{Q}(\alpha)/\mathbb{Q}}(p) = F(0,p)/c_1 = p^{\deg(f)}$ and $N_{\mathbb{Q}(\beta)/\mathbb{Q}}(p) = G(0,p)/c_2 = p^{\deg(g)}$, respectively. Approximately $1/(g_f \cdot g_g)$ of the primes offer a free relation, where g_f and g_g are the orders of the Galois groups of the polynomials f and g, respectively [32]. The free relation $(p, 0)$ is added to the relation table only if all prime ideals of norm p appear in the relation table.

Next, a frequency table is built for all occurring prime ideals which is adjusted as the relation table changes. The relation table is then scanned circularly and relations containing an ideal of frequency 1 (singletons) are removed from it. The program executes as many passes through the table as is needed to remove all singletons.

At the end of the pruning algorithm we would like the remaining number of relations to be larger than the total number of prime ideals. Therefore we need to reserve a surplus of relations for the small prime ideals: Per polynomial, the number of prime ideals below `filtmin` is approximately $\pi(\texttt{filtmin})$, i.e., the number of primes below `filtmin`, see [38]. Consequently, we require a surplus of approximately $(2 - (g_f \cdot g_g)^{-1}) \cdot \pi(\texttt{filtmin})$ relations. If the required surplus is not reached we need to sieve more relations.

Clique algorithm.

If there are sufficiently many more relations than ideals, the user may want to specify how many more relations than large ideals to retain after the pruning stage (`keep`).

In [59, step 3] Pomerance and Smith eject excess relations by simply deleting the heaviest relations. However, as an alternative, they suggest to delete relations which contain many primes of frequency 2. Our approach is similar to this alternative. The algorithm we use is called *clique algorithm*, since it deletes relations that stick together.

Consider the graph with the relations from the relation table as nodes. We connect two nodes if the corresponding relations would be merged in a 2-way merge. The components of the graph are called cliques. The relations in a clique are close to each other in the sense that if one of them is removed, the others will become singletons after some steps and are therefore useless.

The clique algorithm determines all the cliques, evaluates them with the help of a metric and at each step keeps up to a prescribed number of them in a priority heap [35, page 144], ordered by the size of a metric value. The metric being used weighs the contribution from the small prime ideals by adding 1 for each relation in the clique and 0.5 for each free relation. The large prime ideals which occur more than twice in the relation table contribute 0.5^{f-2} where f is the prime ideal's frequency. This way we "penalise" ideals with low frequency. Relation-sets containing many ideals with low frequencies are more likely to be deleted than those containing mainly high frequency ideals. By deleting these low-frequency relation-sets we hope to reduce especially low frequencies even more and get new merge candidates.

Finally, the relations belonging to cliques in the heap are deleted from the relation table. When deleting relations we decrease the ideal frequencies of the primes involved. Singletons may arise and we therefore continue with the singleton processing step. The clique algorithm may be repeated if the number of excess relations does not approximate `keep` sufficiently.

After duplication, singleton and possibly clique processing the relations are read again and only the non-free relations[3] appearing in the relation table are written to the output file. If the input files have grown in the meantime, the new relations are discarded.

3.3.2 Merging

First, we have a closer look at how merging works, which parameters can be given and at how to minimise the weight increase during a k-way merge. Next, we give details about the implementation of the "controlled" merges. Finally we study the influence of merging on Block Lanczos's running time.

Merging aims at reducing the matrix size by combining relations. Throughout this section we give figures about weight changes in the matrix. These figures do not take account of possible other primes that may have been balanced incidentally during the same merge.

[3]Free relations will be generated during the merge stage again.

Parameters `mergelevel`, `maxpass`, `maxrels` and `maxdiscard`.

With the parameter `mergelevel` the user specifies the highest k for which k-way merges are allowed to be executed. The user fixes the maximum number (`maxpass`) of *shrinkage passes* to execute. During a shrinkage pass, all large primes are checked once and possibly merged, see [28, section 7] for more details.

The simplest case is the so-called 2-way merge. A prime ideal I is unbalanced in exactly two relations, r_1 and r_2, and we combine the relations into the relation-set r_1+r_2. As a result, we have one fewer column (r_1 and r_2 disappear, r_1+r_2 enters) as well as one fewer row (prime ideal I) and the total weight has thereby decreased by 2.

In general, if a prime ideal I is unbalanced in exactly k relations ($k \geq 2$),[4] we can choose $k-1$ independent relation pairs out of the possible $\binom{k}{2}$ pairs. For example, if $k=3$, there are 3 possible ways to combine the 3 relations involved, r_1, r_2 and r_3, to a couple, namely r_1+r_2, r_2+r_3 and r_1+r_3. Each one can be obtained from the other two, for instance $r_1+r_3=(r_1+r_2)+(r_2+r_3)$ as all the prime ideals of r_2 are balanced since r_2 appears twice.

After the merge, the prime ideal I is balanced. Its corresponding row has disappeared from the matrix. The total gain of every merge consists in fact in one fewer column and one fewer row. The drawback of merging is, of course, matrix fill-in. A 2-way merge causes no fill-in at all, we even have 2 entries fewer in the matrix. However, a k-way merge, $k \geq 3$, causes the matrix to be heavier by about the weight of $k-2$ relations minus the $2(k-1)$ entries that disappeared.

If the matrix is going to be "lopsided", i.e., if it has many more relations than ideals, it is useful to drop heavy relation-sets. The program therefore discards the ones which contain more relations than the user-determined bound `maxrels`.[5] The user may specify `maxdiscard`, that is, the maximum number of relation-sets to be dropped during one `filter` run. Once `maxdiscard` has been reached, k-way merges, $k \geq 3$, are inhibited.

Minimising the weight increase of a k-way merge.

Which $k-1$ of the possible $\binom{k}{2}$ relation pairs should be chosen in order to achieve the lowest weight increase? First of all, each relation has to appear in at least one relation couple, that is, we need to form independent relation sets, in order not to loose data. Secondly, we focus on minimising the weight increase. In the beginning, when all relations are true single relations, we usually achieve the lowest weight increase by choosing the lightest relation (*pivot*) and combining it with the remaining $k-1$ relations. We call this *pivoting*. More precisely, this happens always when no additional prime ideals except for the prime ideal I become balanced in any of the candidate relation couples. If we assume the

[4]The case $k=1$ denotes a singleton which would be deleted.

[5]We weigh a free relation less than 1 (we used 0.5), because, even if it may have several large primes, it should have less total weight.

pivot relation to be r_k, the weight increase Δw will be exactly

$$\Delta w = (k-2)w(r_k) - 2(k-1). \tag{3.1}$$

The choice becomes more complicated, when additional prime ideals get balanced, especially when we are merging already combined relation-sets. For example, consider the following 5 relations, which are candidates for two 3-way merges with the prime ideals I and J:

$$\begin{aligned}
r_1 &: \ I \text{ and } v-1 \text{ other prime ideals}\\
r_2 &: \ I \text{ and } v-1 \text{ other prime ideals}\\
r_3 &: \ I, J \text{ and } v-2 \text{ other prime ideals}\\
r_4 &: \ J \text{ and } v-1 \text{ other prime ideals}\\
r_5 &: \ J \text{ and } v-1 \text{ other prime ideals}
\end{aligned}$$

For the sake of simplicity, we assume that all the relations have the same weight v and do not share other primes except for I and J. Imagine, r_3 is used as a pivot relation to eliminate I. We get

$$\begin{aligned}
r_1 + r_3 &: \ J \text{ and } 2v-3 \text{ other prime ideals}\\
r_2 + r_3 &: \ J \text{ and } 2v-3 \text{ other prime ideals}\\
r_4 &: \ J \text{ and } v-1 \text{ other prime ideals}\\
r_5 &: \ J \text{ and } v-1 \text{ other prime ideals}
\end{aligned}$$

Now J appears 4 times, so we need a 4-way merge to balance it. For the elimination of J the two relations r_4 and r_5 seem the best pivot candidates in a 4-way merge, since they have lowest weight. However, pivoting with r_5 results into

$$\begin{aligned}
(r_1 + r_3) + r_5 &: \ 3v-4 \text{ prime ideals}\\
(r_2 + r_3) + r_5 &: \ 3v-4 \text{ prime ideals}\\
r_4 + r_5 &: \ 2v-2 \text{ prime ideals}
\end{aligned}$$

with total weight $8v-10$, whereas

$$\begin{aligned}
(r_1 + r_3) + (r_2 + r_3) &: \ 2v-2 \text{ prime ideals}\\
(r_1 + r_3) + r_5 &: \ 3v-4 \text{ prime ideals}\\
r_4 + r_5 &: \ 2v-2 \text{ prime ideals}
\end{aligned}$$

ends with weight $7v-8$.[6] When $v > 2$ we have $8v-10 > 7v-8$ which indicates that we should not stick to pivoting for all the merges.

The problem of minimising the weight increase can be stated using graphs. The vertices are given by the k relations which are candidates for a k-way merge and the $\binom{k}{2}$ edges between them represent possible merges. The edge between two nodes r_i and r_j has weight $w(r_i + r_j)$. Given this weighted graph we wish to select a tree with minimum total weight. The solution is called a *minimum spanning tree* [33, page 460]. This problem is a well-known problem of combinatorial optimisation. In order to solve it we use the algorithm as formulated by Jarník [30, pages 46–47].

[6]The latter situation is also achieved when first using r_1 as a pivot and then doing a 3-way merge with pivot relation r_5.

k	$\left\lfloor \frac{m_{max}}{k-2} 2 \right\rfloor /2$	
	$m_{max} = 7$	$m_{max} = 8$
3	7	8
4	3.5	4
5	2	2.5
6	1.5	2
7	1	1.5
8–9	1	1
10	0.5	1
11–16	0.5	0.5
17–18	—	0.5

Table 3.1: Allowed number of relations in pivot relation-set for k-way merge

Implementation of "controlled" merges.

We limit the weight increase of a single merge by requiring that a merge should not add more than a prescribed number, m_{max}, of original relations to the matrix. We give all the initial relations the same weight (except for free relations that weigh one half), which is reasonable since the relations are the factorisations of numbers of about the same size.

Let us consider k relation-sets which are candidates for a k-way merge. The individual relation-sets may contain several original relations. Suppose the lightest candidate relation-set has j relations, where free relations count for 0.5. Let c be the number of relation-sets with exactly this minimum number j of relations. Shrinkage pass 1 starts with $m = 1$ and we subsequently augment m up until m_{max} and allow for the k-way merge when $(k-2)j \leq m - (c-1)/2$. The m gives the maximum weight increase (in number of relations) allowed during a merge. We introduced c in order to postpone some merges and do the ones where the best way to merge is clear cut first. Since we are still interested in doing lower weight merges before higher weight merges we increase m only every other shrinkage pass and set $c = 1$ during these shrinkage passes. In most of the runs we had $m_{max} = 7$, but we tried $m_{max} = 8$ as well. Solving the inequality $(k-2)j \leq m_{max}$ for k gives $k \leq \frac{m_{max}}{j} + 2$. It follows that, with $m_{max} = 7$, merges with ordinary relations ($j = 1$) are limited to prime ideal frequency 9 whereas free relations ($j = 0.5$) can be used in merges up to prime ideal frequency 16. For the factorisation of RSA-155 we performed merges up to prime ideal frequency 8.

Table 3.1 shows the maximum number of relations a pivot relation-set may consist of, for $m_{max} = 7$ and 8. Even if we are not pivoting, we ask at least one relation not to contain more relations than this bound.

Influence of merging on Block Lanczos's running time.

Given an $m \times n$ matrix, $n > m$, of total weight w, the running time estimate of Block Lanczos is given by $\mathcal{O}(wn) + \mathcal{O}(n^2)$ [49]. Both terms grow with n, so we will focus on reducing n. If we manage to reduce n by a certain factor while w does not grow by more than this factor, we will get a running time reduction, independently of the constants in the two terms. Moreover, we predict the constant in the $\mathcal{O}(n^2)$ term to be the larger one. Therefore, it is natural to write the running time as

$$\mathcal{O}((w + Cn)n) \tag{3.2}$$

with $C \geq 1$. Since we do not need absolute running times, we drop the $\mathcal{O}$-sign and use the function $t(n, w) = (w + Cn)n$. The larger the constant C, the more it will be convenient to reduce the matrix size. The constant depends on the implementation, for example on the number of bits per vector element (K) used.[7] Montgomery (personal communication) at first estimated the constant C to be about 50. For some approximate values of C see Table 3.7 or Table 3.2.

Let us determine a bound for the weight increase Δw such that a merge causing an increase below this bound still is beneficial to the running time. The condition for Δw becomes

$$t(n - 1, w + \Delta w) - t(n, w) < 0. \tag{3.3}$$

Inequality (3.3) is equivalent to

$$0 > n\left((1 - 2n)C - w + (n - 1)\Delta w\right) = (n - 1)(-2Cn - w + n\Delta w) - w - Cn.$$

The inequality is satisfied if $\Delta w < 2C + \frac{w}{n}$. It follows that the allowed weight increase grows with C and the average column weight $\frac{w}{n}$. That means that denser matrices allow heavier merges than sparser matrices do.

Let us calculate a limit for the pivot relation weight j of a general k-way merge, $k \geq 3$. According to equation (3.1) we require

$$\Delta w = (k - 2)j - 2(k - 1) < 2C + \frac{w}{n}.$$

which results into

$$j < \frac{2C + \frac{w}{n} + 2(k - 1)}{k - 2}. \tag{3.4}$$

In Table 3.2 we report the allowed pivot relation weights for merges up to prime ideal frequency 10. We chose $\frac{w}{n} = 30$ (typical after applying only 2- and 3-way merges) and $\frac{w}{n} = 50$ (typical $\frac{w}{n}$ of many of our final matrices). The horizontal lines divide between above and below $\frac{w}{n}$.

[7]Montgomery [49] gives the formula $\mathcal{O}(wn/K) + \mathcal{O}(n^2)$ for the running time.

k	$\left\lceil \frac{2C+\frac{w}{n}+2(k-1)}{k-2} \right\rceil - 1$							
	$\frac{w}{n}=30$				$\frac{w}{n}=50$			
	$C=49$	$C=37$	$C=14$	$C=1$	$C=49$	$C=37$	$C=14$	$C=1$
3	131	107	61	35	151	127	81	55
4	66	54	31	18	76	64	41	28
5	45	37	21	13	51	43	28	19
6	34	28	16	10	39	33	21	15
7	27	23	13	8	31	27	17	12
8	23	19	11	7	26	22	15	10
9	20	17	10	6	23	19	13	9
10	18	15	9	6	20	17	11	8

Table 3.2: Allowed pivot relation weights for k-way merge

From Table 3.2 we can see that 3-way merges can be done with rather heavy pivot relations; even for $C = 1$ and $\frac{w}{n} = 50$ the allowed weight exceeds $\frac{w}{n}$. Denser matrices allow also for denser pivot relations.

By substituting $\frac{w}{n}$ for j in (3.4) we can derive a condition for when to do k-way merges for $k > 3$ with an average weighing pivot relation:

$$\frac{w}{n} < \frac{2C + 2(k-1)}{k-3} \tag{3.5}$$

The analysis for $k = 3$ has to be done separately, we require (3.3) for $\Delta w = \frac{w}{n} - 4$. By reorganising the terms we get $-4(n-1) - \frac{w}{n} - C(2n-1) < 0$ which is always satisfied. This means that 3-way merges with an average weight pivot relation are always profitable, independently from the density of the matrix or the constant C.

Table 3.3 gives the allowed average weights when merging with an average weight pivot relation. If we assume $C < 50$ and we apply the merges in ascending order of prime ideal frequency, 6-way merges with average weighing pivot relations will not be worthwhile because after the 5-way merges we have seen in practice $\frac{w}{n}$ to be around 50, which is higher than the maximum value of 35.

3.4 Other methods in the literature

We would like to mention two articles about similar filter strategies. These are "Solving Large Sparse Linear Systems Over Finite Fields" of LaMacchia and Odlyzko from 1990[36] and "Reduction of Huge, Sparse Matrices over Finite Fields Via Created Catastrophes" of Pomerance and Smith from 1992[59]. Their strategies are similar to each other but differ in some points. Both were designed to reduce the initial data to a substantially smaller matrix. This matrix was allowed to be fairly dense since it was going to be processed by Gaussian

k	$\left\lceil \frac{2C+2(k-1)}{k-3} \right\rceil - 1$			
	$C = 49$	$C = 37$	$C = 14$	$C = 1$
4	103	79	33	7
5	52	40	17	4
6	35	27	12	3
7	27	21	9	3
8	22	17	8	3
9	18	14	7	2
10	16	13	6	2

Table 3.3: Allowed average weights for k-way merge

elimination afterwards. In contrast, the purpose of our method is to reduce the matrix size but still keep it sparse in order to take advantage of the Block Lanczos method. They were dealing with matrices of size up to 300K, we with matrices of size up to 7M. Each reflects the maximum size that could be handled at the time.

Both other methods executed their operations on the matrix itself whereas we dealt with the raw relations. We identified relations with columns in the final matrix whereas they identified relations with rows. Nevertheless, for an easier comparison, we will stick to identify relations with columns in the present description.

They operate only on part of the matrix (active rows) where no fill-in takes place. The operations must be memorised in order to be repeated on the complete matrix afterwards. LaMacchia and Odlyzko store the history in core, whereas Pomerance and Smith keep a history file.

We will distinguish between the pruning and merging step, as in the description of our method. The weight they look at is only the weight of the active primes at that moment.

The pruning step does differ from our approach only in how to delete excess relations. Duplicates and singletons are removed as soon as possible, as in our approach. Pomerance and Smith choose to remove the excess immediately, whereas LaMacchia and Odlyzko remove the excess just before the "collapse" or "catastrophe" during the merge step. Both decide to drop the heaviest relations, but Pomerance and Smith indicate that one might try other strategies (as we did).

In the beginning of the merge stage, a small number of rows (the heaviest, which correspond to small primes) are declared inactive. Merges are done by pivoting with columns that have only one 1 in the active part. There is no fixed limit for the prime ideal frequency up to which to merge. Once all possible merges have been done and there are still 1's in the active part, more rows (again the heaviest) are declared inactive and the merge step is repeated. This is repeated until the active part collapses. This procedure leads to very heavy matrices. To overcome this, LaMacchia and Odlyzko for example, extend the inactive part considerably after it has reached a certain critical size. This way fewer merges can be executed and the fill-in is confined. Nevertheless, the matrices still have high column weights: the lightest example given by LaMacchia and Odlyzko has an average of 115 entries per column for a $6.0 \cdot 10^4$ columns matrix which is much denser than our densest matrix, the $6.3 \cdot 10^6$ columns matrix from Table 3.11 having an average 81 entries per column.[8]

Initially, for a sparse matrix, merges are done with very light columns, since the inactive part is small and cannot contain many 1's. Further on, pivot relations can be very heavy: very probably, the single 1 in the increasingly smaller active part mostly represents a large prime and goes together with many small prime factors, since all polynomial values are about the same size (Pomerance and Smith try to overcome this by also allowing merges with pivot columns having two 1's in the active part of the matrix.). Moreover, they do not make a distinction between "original" pivot relations and already merged ones, which can be substantially heavier.

In our merge procedure we also merge with already merged relations, but this happens in a controlled way. We limit the number of original relations which can be added during a single merge. We also minimise the fill-in per merge by using a minimum spanning tree algorithm instead of the simpler pivoting, see Section 3.3.2. But here we also have to say, that we cannot guarantee to always get the cheapest merge, because we count the contribution from the large prime ideals but only *estimate* the contribution from the small prime ideals.

In 1995, Thomas Denny proposed a Structured Gaussian elimination preliminary step for Block Lanczos [24]. He estimated $C = 1$ for his own Block Lanczos program. We therefore also included $C = 1$ in Tables 3.2 and 3.3.

3.5 Experimental results

The experiments were done with two versions of our program `filter`. Both of them include pruning facilities.

The first version was capable of doing merges up to prime ideal frequency 5 and corresponded to the old program [28, section 7] if invoked with `mergelevel` 2 or 3. With the first version the user needed to specify when to start with the 4- and 5-way merges. For example, in the tables about filter runs (Tables 3.5, 3.8 and 3.10) the notation 4(x) in column `mergelevel` means that 4-way merges started x shrinkage passes after 3-way merges started. 5(x-y) means that 4-way merges started x shrinkage passes after 3-way merges did, and 5-way merges started y shrinkage passes later than 3-way merges.

The present `filter` version does not need this information any more. It can do merges up to prime ideal frequency 18. The merges are done in order

[8]The column weight 70 given in Table 3.11 corresponds to the matrix obtained when dropping the prime ideals of norm below 40.

of weight increase (measured in numbers of original relations). All runs except RSA-155's B6 had $m_{max} = 7$.

Table 3.4 gives an overview of all pruning activities in our experiments for RSA-140, R211 and RSA-155. All the figures are in units of a million. With prime ideals we mean prime ideals above 10M; we need to reserve an excess of 1.3M relations for the small prime ideals. The non-duplicate relation counts differ so much due to the use of different large prime bounds. Apparent errors are due to rounding values to units of one million.

number being factored		RSA-140		R211		RSA-155			
experiment		A	B	A	B	A	B	C	D
raw relations	(1)	65.7	68.5	57.6		130.8			
duplicates	(2)	10.6	11.9	10.6		45.3			
non-duplicates	(3)=(1)−(2)	55.1	56.6	47.0		85.5			
free relations	(4)	0.1	0.1	0.8		0.2			
prime ideals	(5)	54.2	54.7	49.5		78.8			
excess	(6)=(3)+(4)−(5)	1.1	2.0	−1.7		6.9			
singletons	(7)	28.5	28.2	26.5		32.5			
relations left	(8)=(3)+(4)−(7)	26.8	28.5	21.3		53.2			
prime ideals left	(9)	21.5	22.6	18.5		42.6			
excess	(10)=(8)−(9)	5.2	6.0	2.8		10.6			
clique relations	(11)	17.6	18.7	7.4	0	34.1	33.0	29.6	22.9
relations left	(12)=(8)−(11)	9.2	9.8	13.9	21.3	19.1	20.2	23.6	30.3
prime ideals left	(13)	7.8	8.1	12.2	18.5	17.4	18.2	20.6	25.3
excess (=`keep`)	(14)=(12)−(13)	1.4	1.7	1.7	2.8	1.7	2.0	3.0	5.0

Table 3.4: summary of `mergelevel` 0 and 1 runs

The figures in Tables 3.5–3.11 are given in units of a million (M) or a thousand (K). We labelled the experiments with capital letters. All experiments with the same letter started with the same `mergelevel` 1 run.

In Tables 3.5, 3.8 and 3.10, columns 2–6 are input parameters. Column 7–10 are results: column "sets" gives the number of relation-sets remaining after the run, column "discarded" gives the total number of relation-sets which were discarded during the run. "excess" gives how many more relations than the approximate total number of ideals we retained. It indicates how many more relations we might still throw away in a further run. "not merged" gives the number of large prime ideals of frequency smaller or equal to `mergelevel` among the output relations. For the runs with the new version we also report the number of output relation-sets made of one single relation since among those could be candidates for future high-way merges.

The Block Lanczos code typically finds almost K dependencies [49], where K is the number of bits per vector element. This enables us to drop the heaviest rows which leads to substantially lighter matrices.[9] We dropped the rows

[9]In particular, all quadratic character rows are omitted. The pseudo-dependencies being found for this reduced matrix must be combined to real dependencies afterwards.

corresponding to prime ideals of norm smaller than 50 for R211, whereas for RSA-140 and RSA-155, which have both exceptionally many small prime ideals, we omitted the prime ideals of norm smaller than 40.[10] In addition, the Block Lanczos code truncates every $m \times n$ matrix by default to $m \times (m + K + 100)$.

The tables featuring matrix data (Tables 3.6, 3.9 and 3.11) are made of two parts. In the first part we state the real size $(m \times n)$, weight (w) and average column weight $(\frac{w}{n})$ of the matrices built. The numbers between two lines express the changes in size (number of columns) and weight from one matrix to the smaller one as percentages. Note that a $i\%$ decrease in matrix size makes the term wn shrink as long as the weight does not increase by more than $\frac{100i}{100-i}\%$ which is slightly larger than $i\%$. The second part shows the effective weight (w_{eff}) after truncating the matrix to size $m \times (m + K + 100)$, the effective average column weight $(\frac{w_{\mathit{eff}}}{m+K+100})$ and the Block Lanczos timings from a Cray C90 and a Silicon Graphics Origin 2000. The timings can vary substantially according to the load on the machines (other jobs interacting with ours): time differences of 20% are not unusual. Aiming at a fair comparison we tried to run the matrices at times with comparable load. In our tables, comparable timings are written in the same column. Only one Block Lanczos job per number was completely executed. All times in the tables are extrapolations: we did a short run, took the time of the fastest iteration and multiplied it by the number of iterations $(m + K + 100)/(K - 0.76)$, see [49].

3.5.1 RSA-140

This 140-digit number was factored on February 2, 1999. The experiment series A started with 65.7M raw relations, B with 68.5M from 5 different sites. We removed 1.4M and 1.6M duplicates, respectively, with `mergelevel` 0 runs on each contributor's data. The experiments in Table 3.5 start with the remaining 64.3M respectively 66.9M relations having 54.2M and 54.7M large prime ideals, respectively. After the pruning step (with `filtmin`= 10M) we need an excess of $\frac{239}{120}\pi(10\text{M}) = 1.3\text{M}$ for the small prime ideals. For a summary of `mergelevel` 0 and 1 runs, see Table 3.4.

In this paragraph we only describe experiment series A. The `mergelevel` 1 run on the whole bunch of data removed another 9.2M duplicates and added 0.1M free relations for large primes. Note, that at this point the excess 64.3M − 54.2M − 9.2M + 0.1M = 1.1M[11] was less than the needed 1.3M. The excess was sufficient only after removing the singletons, when we were left with 26.8M relations having 21.5M large prime ideals. The clique algorithm removed a total of 17.6M relations to approximate the excess of 1.4M = 9.2M − 7.8M.

The factorisation was done using matrix A1.1 which took 100h on the Cray.

[10]These figures match with the implementation for $K = 64$. For $K = 128$, we could even have dropped the prime ideals up to norm 180. The resulting lighter matrices would have led to shorter timings for that implementation. However, for simplicity, we used the same matrices for both the $K = 64$ and the $K = 128$ versions.

[11]The apparent arithmetical error is due to rounding all numbers to units of a million.

Only 2- and 3-way merges were performed, because the code for higher than 3-way merges was not ready by then. For logistic reasons we had built the matrix before we received all the data.

With the complete data (experiment series B) the excess was enough from the beginning. Furthermore, a matrix constructed from this data by applying the same filter strategy as for A1.1 would have performed better than A1.1 as one can imagine when comparing A1.1.2.1 to B1.2: both did merges up to prime ideal frequency 5 and the latter is smaller in size *and* weight.

We also tried `mergelevel` 8 (B2) with $m_{max} = 7$ which was introduced only just before the factorisation of RSA-155. The program stopped with k-way merges, $k \geq 3$ at shrinkage pass 10 after having deleted 381K relations. This means that only merges with a maximum weight increase of 6 original relations had been done. Matrix B2 beats the `mergelevel` 5 matrix of the same series (B1.2).

In Table 3.6 one can see from the percentages that each size reduction should have a favourable effect on Block Lanczos's running time which is confirmed by the time column.

These experiments confirm our idea of the advantage of higher-way merges. They show that collecting more data than necessary is recommendable. It does not become clear, however, how much excess data one should keep after the pruning step.

experiment	`mergelevel`	`filtmin`	`maxdiscard`	`maxrels`	`maxpass`	sets	discarded	excess	not merged
A	1	10M	`keep` 1.4M			9.2M	46 040K	90K	-
A1	2	10M	-	4.0	6	6.0M	54K	36K	59
A1.1	3	10M	unlim.	10.0	10	4.7M	3K	33K	0
A1.1.1	4(0)	10M	20K	10.0	10	4.2M	20K	13K	243K
A1.1.2	4(0)	10M	20K	12.0	10	4.0M	14K	20K	0
A1.1.3	4(0)	10M	20K	11.0	10	4.0M	20K	13K	48K
A1.1.2.1	5(0-0)	8M	17K	15.0	10	3.5M	17K	4K	0
B	1	10M	`keep` 1.7M			9.8M	46 906K	384K	-
B1	4(5)	10M	300K	8.0	12	4.3M	170K	208K	6K
B1.1	5(1-3)	10M	200K	11.5	10	3.6M	85K	128K	1K
B1.2	5(1-3)	10M	200K	10.5	10	3.4M	200K	14K	28K
B2	8	10M	375K	8.0	15	3.3M	383K	1K	909K/455K

Table 3.5: RSA-140 filter runs

With each timing column, we fitted a surface $t = s_1 n^2 + s_2 nw$ to the points (n, w, t). The fits were done by `gnuplot`'s implementation of the nonlinear least-squares (NLLS) Marquardt-Levenberg algorithm. The quotient s_1/s_2 corresponds to the C from (3.2). Table 3.7 gives some possible values for C.

exp.	matrix size	%	weight	%	col.w.	w_{eff}	col.w.	Cray	SGI [a]	
A1.1	4 671K × 4 704K		151.1M		32.1	147.4M	31.5	75h	59d	24d
A1.1.1	4 180K × 4 193K	−11	163.1M	+8	38.9	161.3M	38.6	65h	56d	22d
A1.1.3	3 999K × 4 012K	−4	168.7M	+3	42.0	166.8M	41.7	63h	54d	21d
A1.1.2	3 960K × 3 980K	−1	171.1M	+1	43.0	168.1M	42.4	62h	53d	20d
A1.1.2.1	3 504K × 3 507K	−12	191.3M	+12	54.5	190.8M	54.4	56h	51d	18d
B1.2	3 380K × 3 394K		178.8M		52.7	176.8M	52.3	51h	46d	16d
B2	3 285K × 3 286K	−3	182.1M	+2	55.4	182.0M	55.4	50h	43d	15d

Table 3.6: RSA-140 matrices

[a]The second column gives timings from the $K = 128$ implementation.

Block Lanczos implementation	s_1		s_2		C	
vectorised Cray code with $K = 64$	1.84	±0.06	0.0499	±0.0014	37	±2
SGI code with $K = 64$	0.86	±0.14	0.060	±0.003	14	±3
improved SGI code with $K = 128$[a]	0.69	±0.08	0.0140	±0.0018	49	±12

Table 3.7: C values for different Block Lanczos implementations

[a]This version ‘under development’ by Montgomery is being optimised for cache usage rather than vectorisation. It is being redesigned to allow parallelisation, but we used only one processor.

$C = 14$ is much smaller than we had initially expected. According to Table 3.2, with $C = 14$ and assuming $\frac{w}{n} = 30$ we have that 4-way merges are convenient with pivot relations up to weight 31, which is slightly above average whereas 5-way merges should be done with lighter than average (max. 21 entries) pivot relations. When assuming $\frac{w}{n} = 50$ the maxima are higher but below average also for 4-way merges.

Why then did the matrices, which were constructed by more or less brutally doing all possible 3-, 4- and 5-way merges,[12] perform better than we would expect from looking at the figures in Table 3.3 and 3.2? It seems most merges were able to find a pivot relation with much smaller weight than average. Furthermore, we must consider that the inequalities (3.4) and (3.5) do not take account of the weight and size reduction obtained by discarding relation-sets which are made of more than `maxrels` relations. Some benefit also comes from the minimum spanning tree algorithm.

With $C = 49$ and $\frac{w}{n} = 30$, even above average 6-way merges can be beneficial.

[12]For A1.1.2.1, all possible merges up to prime ideal frequency 5, for prime ideals of norm larger than 8M, had been performed.

3.5.2 R211

The following two tables give data concerning filter experiments with the special 211-digit number R211:= $(10^{211}-1)/9$, which is a so-called "repunit", since all its digits are 1. It was factored on April 8, 1999. Five sites produced a total of 57.6M raw relations. 1.2M duplicates were removed during `mergelevel` 0 runs on the individual data. The experiment series A and B both started with the remaining 56.4M relations having 49.5M prime ideals of norm above 10M. This means that we had 6.9M more relations than prime ideals which seemed to be enough since we needed to reserve $\frac{23}{12}\pi(10\text{M}) = 1.3\text{M}$ more relations accounting for the small prime ideals. Unfortunately, the `mergelevel` 1 run on the complete data set revealed 9.4M duplicates. The remaining 47.0M relations plus 0.8M free relations were *less* than the number of prime ideals. However, we did not need to sieve further since we had an excess after removing the 26.5M singletons. The clique algorithm started hence with 21.3M relations having 18.6M prime ideals of norm larger than 10M, which is an excess of 2.8M. See Table 3.4.

Experiment series A gives the parameters and results of the `filter` runs that led to the matrix that was used to factor the number; it took 120 hours on the Cray. B shows a different approach, where we kept 1.1M more relations than for A after the pruning step, leaving more choice for merging.

experiment	`mergelevel`	`filtmin`	`maxdiscard`	`maxrels`	`maxpass`	sets	discarded	excess	not merged
A	1	10M	`keep` 1.7M			13.9M	33 839K	433K	-
A1	4(5)	20M	300K	6.0	10	6.8M	304K	124K	1 637K
A1.1	5(5-10)	20M	15K	12.0	15	5.6M	15K	109K	796K
A1.1.1[a]	5(5-10)	8M	50K	15.0	15	4.9M	n.a.	63K	n.a.
B	1	10M	`keep` 2.8M			21.3M	26 488K	1 484K	-
B1	4(5)	20M	1 300K	6.0	10	6.7M	1 310K	206K	1 410K
B1.1	5(5-10)	20M	170K	12.0	15	4.8M	170K	11K	97K
B1.1.1	5(1-3)	8M	10K	18.0	10	4.6M	4K	8K	2
B2	8	10M	1 400K	9.0	15	4.7M	1 421K	30K	1 244K/925K
B3	8	10M	1 400K	10.0	15	4.5M	1 423K	64K	918K/777K

Table 3.8: R211 filter runs

[a]This run was done with the flag `regroup`, which splits up existing relation-sets and does merges from scratch, which leads to different relation-sets.

Both `mergelevel` 4 runs can actually be considered `mergelevel` 3 runs, since the maximum number of discards, `maxdiscard`, was reached before 4-way merges would have started.

Experiment series B achieved smaller matrices than A. The reason must be the different `keep` values during the pruning stage. Experiment series A kicked out 7.4M relations with the clique algorithm whereas B kept all the

exp.	matrix size	%	weight	%	col.w.	w_{eff}	col.w.	Cray				SGI
A1.1.1	4 820K × 4 896K		234.2M		47.8	221.2M	45.88	118h	-	97h	93h	96d
B1.1	4 863K × 4 877K	−0	223.3M	−5	45.8	221.3M	45.92	119h	-	97h	95h	97d
B2	4 723K × 4 754K	−3	231.9M	+4	48.8	228.2M	49.10	-	95h	93h	92h	95d
B1.1.1	4 661K × 4 670K	−2	231.2M	−0	49.5	229.3M	49.60	115h	-	93h	91h	95d
B3	4 503K × 4 569K	−2	247.5M	+7	54.2	239.0M	53.06	-	90h	-	-	-

Table 3.9: R211 matrices

excess relations, performed more merges and discarded more relations during the merge steps. We can conclude that for this data the best thing was to skip the clique algorithm. This is strongly connected to the fact that we barely had enough relations. Sieving any longer would surely have led to smaller matrices.

Matrix A1.1.1 performed better than matrix B1.1, which may seem counter-intuitive since B1.1 produced the smaller and lighter matrix. However, matrix A1.1.1 contained fewer rows (fewer prime ideals) than matrix B1.1 and due to the default truncation taking place in the Block Lanczos algorithm the effective A1.1.1 matrix was smaller in size and weight than the effective B1.1 matrix.

At B2 we also tried `mergelevel` 8 while having $m_{max} = 7$. `maxdiscard` was reached already at shrinkage pass 9 (with 15 possible passes) when the allowed weight increase was 5 original relations. The final matrix was larger than B1.1.1. We had chosen `maxrels` too low. It was 9, compared to 18 in B1.1.1. With `maxrels` 10 we achieved the desired reduction (B3).

3.5.3 RSA-155

The 155-digit number RSA-155 (512 bits!) was factored on August 22, 1999. A total of 130.8M relations were collected from 12 different sites. 6.1M relations were removed in individual `mergelevel` 0 runs. Another 39.2M duplicates where removed in a `mergelevel` 0 run on the whole amount of data. All the experiments below started with the remaining 85.5M relations and its 0.2M free relations. Therefore, in contrast to the previous examples, the figures in the discarded column do not contain any duplicates. See Table 3.4 for details.

Matrix B2 was used for the factorisation. It took 225 hours on the Cray.

The experiments indicate that retaining more data (`keep` $\geq$ 3.0M) after the pruning stage did not help to reduce the size of the matrix.

Experiments B4 and D1 discarded too many relation-sets which is recognisable from the negative excess.

In B2 merging was stopped at shrinkage pass 11, while $m = 6$. Since there were still many unmerged ideals in B2, we tried to make the matrix smaller by increasing `maxrels` in B3 which allows also relation-sets with 10 relations, which were deleted in test B2. But even after this run many potential merge candidates remained unmerged, although `maxdiscard` was not reached. This indicates that the weight increase of the merges was considered too high and the merges were subsequently not executed. Next, we tried `mergelevel` 16, which

experiment	`mergelevel`	`filtmin`	`maxdiscard`	`maxrels`	`maxpass`	sets	discarded	excess	not merged
A	1	10M	`keep` 1.7M			19.1M	66 593K	385K	-
A1	5(1-3)	10M	370K	11.0	12	7.1M	370K	15K	67K
B	1	10M	`keep` 2.0M			20.2M	65 531K	684K	-
B1	8	10M	600K	9.0	15	6.9M	603K	81K	1 611K/764K
B2	8	7M	670K	9.0	15	6.7M	672K	13K	1 576K/716K
B3	8	7M	670K	10.0	15	7.1M	366K	317K	1 432K/744K
B4	16	7M	670K	9.0	15	6.6M	690K	−5K	4 130K/694K
B5	16	7M	670K	10.0	15	6.8M	482K	193K	3 797K/562K
B6	18	7M	670K	10.0	15	6.3M	672K	n.a.	n.a.
C	1	10M	`keep` 3.0M			23.6M	62 092K	1 682K	-
C1	8	10M	1 670K	8.0	15	6.8M	1 675K	7K	1 710K/698K
D	1	10M	`keep` 5.0M			30.3K	55 402K	3 677K	-
D1	8	10M	3 670K	7.0	15	7.1M	3 698K	−20K	2 118K/780K

Table 3.10: RSA-155 filter runs

is the maximum prime ideal frequency you can have a merge with for $m_{max} = 7$. Some reduction was achieved (B4 and B5). Finally, we took $m_{max} = 8$ together with **`mergelevel`** 18 and **`maxrels`** 10. **`maxdiscard`** was reached during shrinkage pass 14, when $m = m_{max}$.

exp.	matrix size	%	weight	%	col.w.	w_{eff}	col.w.	Cray
B2	6 699K × 6 711K	−5	417.1M	+7	62.2	415.5M	62.0	218h
B6	6 342K × 6 354K		445.3M		70.1	443.4M	69.9	213h

Table 3.11: RSA-155 matrices

Matrix B6 is 5% smaller than B2 but also 7% heavier. With $C = 14$ we can expect to save $1 - \frac{14 \cdot 6.342^2 + 6.342 \cdot 445.3}{14 \cdot 6.699^2 + 6.699 \cdot 417.1} \approx 1\%$ running time, which is too small a gain to accept the weight increase, whereas with $C = 37$ or $C = 49$ we may save 3% or 4%, respectively. The effective runs on the Cray ($C = 37$) indicate a saving of 2%.

3.6 Conclusions

We extended our previous `filter` program to allow higher-way merges and proved theoretically and practically that we can reduce Block Lanczos running time by performing higher-way merges. We determined limits for the weight of pivot columns.

During a merge, instead of merging by pivoting we calculate a minimum spanning tree in order to assure minimum weight increase.

A denser matrix allows for more weight increase during a merge than a lighter one: this means we can merge with denser pivot columns. Therefore we do the light merges before the heavier ones.

We determined the ratio between the two terms characterising the running time of Block Lanczos for different implementations. To which extent we can profit from higher-way merges depends on this ratio. We saw values ranging from 14 to 49. With the help of this constants we can estimate the running time of a matrix, given the running time of another matrix.

Collecting more data than necessary is advisable. The clique algorithm enables us to get rid of excess data quickly and in a sensible way. It is a useful tool when having abundant excess.

Appendix A

Factorisation of RSA-140 using the Number Field Sieve

Stefania Cavallar, CWI, Amsterdam, The Netherlands,
Bruce Dodson, Lehigh University, Bethlehem, PA, USA,
Arjen Lenstra, Citibank, Parsippany, NJ, USA,
Paul Leyland, Microsoft Research Ltd., Cambridge, UK,
Walter Lioen, CWI, Amsterdam, The Netherlands,
Peter L. Montgomery, Microsoft Research, USA and CWI, Amsterdam, The Netherlands,
Brian Murphy, The Australian National University, Canberra, Australia,
Herman te Riele, CWI, Amsterdam, The Netherlands and
Paul Zimmermann, Inria Lorraine and Loria, Nancy, France.

Abstract

On February 2, 1999, we completed the factorisation of the 140-digit number RSA-140 with the help of the Number Field Sieve factoring method (NFS). This is a new general factoring record.[2] The previous record was established on April 10, 1996 by the factorisation of the 130-digit number RSA-130, also with the help of NFS. The amount of computing time spent on RSA-140 was roughly twice that needed for RSA-130, about half of what could be expected from a straightforward extrapolation of the computing time spent on factoring RSA-130. The speed-up can be

This chapter is a slight revision of the article "Factorization of RSA-140 using the number field sieve" which appeared in the proceedings of the ASIACRYPT'99 conference [16].

[2]This record was broken in the meantime with the factorisation of RSA-155 (see Appendix B) in August 1999 and of a 158 digit number [6] in January 2002.

attributed to a new polynomial selection method for NFS which will be sketched in this paper.

The implications of the new polynomial selection method for factoring a 512-bit RSA modulus are discussed and it is concluded that 512-bit (= 155-digit) RSA moduli are easily and realistically within reach of factoring efforts similar to the one presented here.

A.1 Introduction

Factoring large numbers is an old and fascinating métier in number theory which has become important for cryptographic applications after the birth, in 1977, of the public-key cryptosystem RSA [64]. Since then, people have started to keep track of the largest (difficult) numbers factored so far, and reports of new records were invariably presented at cryptographic conferences. We mention EUROCRYPT '89 (C100[3] [41]), EUROCRYPT '90 (C107 and C116 [42]), CRYPTO '93 (C120, [23]), ASIACRYPT '94 (C129, [3]) and ASIACRYPT '96 (C130, [20]). The 130-digit number was factored with help of the *Number Field Sieve* method (NFS), the others were factored using the *Quadratic Sieve* method (QS).

For information about QS, see [58]. For information about NFS, see [39]. For additional information, implementations and previous large NFS factorisations, see [26, 27, 28, 29].

In this paper, we report on the factoring of RSA-140 by NFS and the implications for RSA. The number RSA-140 was taken from the RSA Challenge list [65]. In Section A.2 we estimate how far we are now from factoring a 512-bit RSA modulus. In Section A.3, we sketch the new polynomial selection method for NFS and we give the details of our computations which resulted in the factorisation of RSA-140.

A.2 How far are we from factoring a 512-bit RSA modulus?

RSA is widely used today. We quote from RSA Laboratories' "Frequently Asked Questions about today's Cryptography 4.0"(see [66] for version 4.1):

> Is RSA currently in use?
>
> RSA is currently used in a wide variety of products, platforms, and industries around the world. It is found in many commercial software products and is planned to be in many more. RSA is built into current operating systems by Microsoft, Apple, Sun, and Novell. In hardware, RSA can be found in secure telephones, on Ethernet network cards, and on smart cards. In addition, RSA is incorporated

[3]By "Cxxx" we denote a composite number having xxx decimal digits.

> into all of the major protocols for secure Internet communications, including S/MIME (see Question 5.1.1), SSL (see Question 5.1.2), and S/WAN (see Question 5.1.3). It is also used internally in many institutions, including branches of the U.S. government, major corporations, national laboratories, and universities.
>
> At the time of this publication, RSA technology is licensed by about 350 companies. The estimated installed base of RSA encryption engines is around 300 million, making it by far the most widely used public-key cryptosystem in the world. This figure is expected to grow rapidly as the Internet and the World Wide Web expand.

The best size for an RSA key depends on the security needs of the user and on how long the data needs to be protected. At present, information of very high value is protected by 512-bit RSA keys. For example, CREST [21] is a system developed by the Bank of England and used to register all the transfers of stocks and shares listed in the United Kingdom. The transactions used to be protected using 512-bit RSA keys at the time of the writing of this paper. Allegedly, 512-bit RSA keys protect 95% of today's E-commerce on the Internet [68].

The amount of CPU time spent to factor RSA-140 is estimated to be only twice that used for the factorisation of RSA-130, whereas on the basis of the heuristic complexity formula [13] for factoring large N by NFS:

$$O\left(\exp\left((1.923 + o(1))(\log N)^{1/3}(\log\log N)^{2/3}\right)\right),$$

one would expect an increase in the computing time by a factor close to four. This has been made possible by algorithmic improvements (mainly in the polynomial generation step [50], and to a lesser extent in the sieving step and the filter step of NFS), and by the relative increase in memory speed of the workstations and PCs used in this project.

After the completion of RSA-140, we completely factored the 211-digit number $10^{211} - 1$ with the Special Number Field Sieve (SNFS) at the expense of slightly more computational effort than we needed for RSA-140. We notice that the polynomial selection stage is easy for $10^{211} - 1$. Calendar time was about two months. This result means a new factoring record for SNFS [61]. The previous SNFS record was the 186-digit number $32633^{41} - 1$ (see [60]).

Experiments indicate that the approach used for the factorisation of RSA-140 may be applied to RSA-155 as well. Estimates based on these experiments suggest that the total effort involved in a 512-bit factorisation (RSA-155 is a 512-bit number) would require only a fraction of the computing time that has been estimated in the literature so far. Also, there is every reason to expect that the matrix size, until quite recently believed to be the main stumbling block for a 512-bit factorisation using NFS, will turn out to be quite manageable. As a result 512-bit RSA moduli do, in our opinion, not offer more than marginal security, and should no longer be used in any serious application.

A.3 Factoring RSA-140

We assume that the reader is familiar with NFS [39], but for convenience we briefly describe the method here. Let N be the number we wish to factor, known to be composite. There are four main steps in NFS: polynomial selection, sieving, linear algebra, and square root.

In the *polynomial selection step*, two irreducible polynomials $f_1(x)$ and $f_2(x)$ with a common root m mod N are selected having as many as practically possible smooth values over a given factor base.

In the *sieving step* which is by far the most time-consuming step of NFS, pairs (a, b) are found with $\gcd(a, b) = 1$ such that both

$$b^{\deg(f_1)} f_1(a/b) \text{ and } b^{\deg(f_2)} f_2(a/b)$$

are smooth over given factor bases, i.e., factor completely over the factor bases. Such a pair (a, b) is called a *relation.* The purpose of this step is to collect so many relations that several subsets S of them can be found with the property that a product taken over S yields an expression of the form

$$X^2 \equiv Y^2 \pmod{N}. \tag{A.1}$$

For approximately half of these subsets, computing $\gcd(X - Y, N)$ yields a non-trivial factor of N (if N has exactly two distinct factors).

In the *linear algebra step*, the relations found are first filtered with the purpose of eliminating duplicate relations and relations in which a prime or prime ideal occurs which does not occur in any other relation. If a prime ideal occurs in exactly two or three relations, these relations are combined into one or two (respectively) so-called relation-sets. These relation-sets form the columns of a very large sparse matrix over $\mathcal{F}_2$. With help of an iterative block Lanczos algorithm a few dependencies are found in this matrix. This is the main and most time- and space-consuming part of the linear algebra step.

In the *square root step*, the square root of an algebraic number of the form

$$\prod_{(a,b)\in S} (a - b\alpha)$$

is computed, where α is a root of one of the polynomials $f_1(x), f_2(x)$, and where a, b and the cardinality of the set S are all a few million. The norms of all $(a - b\alpha)$'s are smooth. This leads to a congruence of the form (A.1).

In the next four subsections, we describe these four steps, as carried out for the factorisation of RSA-140. We pay most attention to the polynomial selection step because, here, new ideas have been incorporated which led to a reduction of the expected – and actual – sieving time for RSA-140 (extrapolated from the RSA-130 sieving time) by a factor of 2.

A.3.1 Polynomial selection

For number field sieve factorisations we use two polynomials $f_1, f_2 \in \mathbb{Z}[x]$ with, amongst other things, a common root m mod N. For integers as large as RSA-140, a modified base-m method is the best method we know of choosing these polynomials. Montgomery's "two-quadratics" method [28] is the only known alternative, and it is unsuitable for numbers this large. With the base-m method, we fix a degree d (here $d = 5$) then seek $m \approx N^{1/(d+1)}$ and a polynomial f_1 of degree d for which

$$f_1(m) \equiv 0 \pmod{N}. \tag{A.2}$$

The polynomial f_1 descends from the base-m representation of N. Indeed, we begin with $f_1(x) = \sum_{i=0}^{d} a_i x^i$ where the a_i are the coefficients of the base-m representation, adjusted so that $-m/2 \le a_i < m/2$.

Sieving occurs over the homogeneous polynomials $F_1(x,y) = y^d f_1(x/y)$ and $F_2(x,y) = x - my$. The aim for polynomial selection is to choose f_1 and m such that the values $F_1(a,b)$ and $F_2(a,b)$ are simultaneously smooth at many coprime integer pairs (a,b) in the sieving region.

We consider this problem in two stages; first we must decide what to look for, then we must decide how to look for it. The first stage requires some understanding of polynomial yield; the second requires techniques for generating polynomials with good yield. In this paper we seek only to outline our techniques. Full details will be published at a later date.

Polynomial yield.

The *yield* of a polynomial $F(x,y)$ refers to the number of smooth (or almost smooth) values it produces in its sieve region. Ultimately of course we seek a *pair* of polynomials F_1, F_2 with good yield. Since F_2 is linear, all primes are roots of F_2, so the difficult polynomial is the non-linear F_1. Hence, initially, we speak only of the yield of F_1.

There are two factors which influence the yield of F_1. These are discussed in a preliminary manner in [53]. We call the factors *size* and *root properties.* Choosing good F_1 requires choosing F_1 with a good *combination* of size and root properties.

By *size* we refer to the magnitude of the values taken by F_1. It has always been well understood that size affects the yield of F_1. Indeed previous approaches to polynomial selection have sought polynomials whose size is smallest (for example, [20]).

The influence of root properties however, has not previously been either well understood or adequately exploited. By *root properties* we refer to the extent to which the distribution of the roots of F_1 modulo small p^k, for p prime and $k \ge 1$, affects the likelihood of F_1 values being smooth. In short, if F_1 has many roots modulo small p^k, the values taken by F_1 "behave" as if they are much smaller than they actually are. That is, on average, the likelihood of F_1-values being smooth is increased. We are able to exploit this property to the extent that F_1

values behave as if they are as little as 1/1000 their actual value. We estimate this property alone increases yield by a factor of four due (by comparison to sieving over random integers of the same size).

Generating polynomials with good yield.

We consider this problem in two stages. In the first stage we generate a large sample of good polynomials. Although each polynomial generated has a good combination of size and root properties, there remains significant variation in the yield across the sample. Moreover, there are still far too many polynomials to conduct sieving experiments on each one. Thus in the second stage we identify *without* sieving, the best polynomials in the sample. The few polynomials surviving this process are then subjected to sieving experiments.

Consider the first stage. We concentrate on so-called *skewed* polynomials, that is, polynomials whose first few coefficients (a_5, a_4 and a_3) are small compared to m, and whose last few coefficients (a_2, a_1 and a_0) may be large compared to m. In fact usually $|a_5| < |a_4| < \ldots < |a_0|$. To compensate for the last few coefficients being large, we sieve over a region much longer in x than y. We take the region to be a rectangle whose length-to-height ratio is s.

Notice that any base-m polynomial may be re-written so that sieving occurs over a rectangle of skewness s. Let $m = O(N^{1/(d+1)})$ giving an unmodified base-m polynomial F_1 with coefficients also $O(N^{1/(d+1)})$. The expected sieve region for F_1 is a "square" given by $\{(x, y) : -M \leq x \leq M \text{ and } 1 \leq y \leq M\}$ for some M. For some (possibly non-integer) $s \in \mathbb{R}$ let $x' = x/\sqrt{s}$, $y' = y\sqrt{s}$ and $m' = ms$. The polynomials $F_1(x', y')$ and $F_2(x', y')$ with common root m', considered over a rectangle of skewness s and area $2M^2$, have the same norms as F_1 and F_2 over the original square region. Such a skewing process can be worthwhile to increase the efficiency of sieving.

However, we have additional methods for constructing *highly* skewed polynomials with good yields. Hence, beyond simply skewing the region on unmodified base-m polynomials, we focus on polynomials which are themselves intrinsically skewed. The search begins by isolating skewed polynomials which are unusually small over a rectangle of some skewness s and which have better than average root properties. The first quality comes from a numerical optimisation procedure which fits a sieve region to each polynomial. The second quality comes from choosing (small) leading coefficients divisible by many small p^k.

We then exploit the skewness to seek adjustments to f_1 which cause it to have exceptionally good root properties, *without* destroying the qualities mentioned above. We can make any adjustment to f_1 as long as we preserve (A.2). We make what we call a *rotation* by P for some polynomial $P(x)$. That is, we let

$$f_{1,P}(x) = f_1(x) + P(x) \cdot (x - m)$$

where $P \in \mathbb{Z}[x]$ has degree small compared to d. Presently we use only linear $P(x) = j_1 x - j_0$ with j_1 and j_0 small compared to a_2 and a_1 respectively. We use a sieve-like procedure to identify pairs (j_1, j_0) which cause $f_{1,P}$ to have exceptionally good root properties mod small p^k. At the end of this procedure (with $p^k < 1000$ say) we have a large set of candidate polynomials.

Consider then the second stage of the process, where we isolate without sieving the polynomials with highest yield. Notice that as a result of looking at a large range of a_d the values of m may vary significantly across the sample. At this stage it is crucial then to consider *both* F_1 and F_2 in the rating procedure. Indeed, the values s vary across the sample too.

We use a quantitative estimate of the effect of the root properties of each polynomial. We factor this parameter into estimates of smoothness probabilities for F_1 and F_2 across a region of skewness s. It is not necessary to estimate the yield across the region, simply to rank the polynomial pairs in the order in which we expect their yields to appear. Of course to avoid missing good polynomial pairs it is crucial that the metric so obtained be reliable.

At the conclusion of this procedure we perform short sieving experiments on the top-ranked candidates.

Results.

Before discussing the RSA-140 polynomial selection results, we briefly consider the previous general factoring record, RSA-130 [20]. As a test, we repeated the search for RSA-130 polynomials and compared our findings to the polynomial used for the factorisation. We searched for non-skewed polynomials only, since that is what was used for the RSA-130 factorisation. Despite therefore finding fewer polynomials with exceptional root properties, we did, in a tiny fraction of the time spent on the RSA-130 polynomial search, find several small polynomials with good root properties. Our best RSA-130 polynomial has a yield approximately twice that of the polynomial used for the factorisation. In essence, this demonstrates the benefit of knowing "what to look for".

The RSA-140 search however, further demonstrates the benefit of knowing "how to look for it". Here of course we exploit the skewness of the polynomials to obtain exceptional root properties.

Sieving experiments on the top RSA-140 candidates were conducted at CWI using line sieving. All pairs were sieved over regions of the same area, but skewed appropriately for each pair. Table 1 shows the relative yields of the top five candidate pairs, labelled A, ..., E. These yields match closely the predictions of our pre-sieving yield estimate.

Poly.	Rel. Yield
A	1.00
B	0.965
C	0.957
D	0.931
E	0.930

Table A.1: Relative Yields of the top RSA-140 polynomials

The chosen pair, pair A, is the following:

$$\begin{aligned} F_1(x,y) \;=\; & 43\,96820\,82840 & x^5 \\ & +39031\,56785\,38960 & y\,x^4 \\ & -7387\,32529\,38929\,94572 & y^2x^3 \\ & -190\,27153\,24374\,29887\,14824 & y^3x^2 \\ & -6\,34410\,25694\,46461\,79139\,30613 & y^4x \\ & +31855\,39170\,71474\,35039\,22235\,07494 & y^5 \end{aligned}$$

and

$$F_2(x,y) = x - 3\,44356\,57809\,24253\,69517\,79007\;\; y,$$

with $s \approx 4000$.

Consider F_1, F_2 with respect to size. We denote by a_{max} the largest $|a_i|$ for $i = 0, \ldots, d$. The un-skewed analogue, $F_1(63x,\, y/63)$, of $F_1(x,y)$ has

$$a_{max} \approx 5 \cdot 10^{20}.$$

A typical unmodified base-m polynomial has

$$a_{max} \approx 1/2N^{1/6} \approx 8 \cdot 10^{22}.$$

The un-skewed analogue, $F_2(63x,\, y/63)$, of $F_2(x,y)$ has

$$a_{max} \approx 3N^{1/6}.$$

Hence, compared to the typical case F_1 values have shrunk by a factor about 160 whilst F_2 values have grown by a factor of 3. F_1 has real roots x/y near -4936, 2414, and 4633.

Now consider F_1 with respect to root properties. Notice that a_5 factors as $2^3 \cdot 3^2 \cdot 5 \cdot 7 \cdot 11 \cdot 13 \cdot 41 \cdot 29759$. Since also $4|a_4$ and $2|a_3$, $F_1(x,y)$ is divisible by 8 whenever y is even. $F_1(x,y)$ has at least three roots x/y modulo each prime from 3 to 17 (some of which are due to the factorisation of the leading coefficient), and an additional 35 such roots modulo the 18 primes from 19 to 97.

We estimate that the yield of the pair F_1, F_2 is approximately eight times that of a skewed pair of average yield. Approximately a factor of four in that eight is due to the root properties, the rest to its size. We estimate the effort spent on the polynomial selection to be equivalent to 0.23 CPU years (approximately 60 MIPS-years). Searching longer may well have produced better polynomials, but we truncated the search to make use of idle time on workstations over the Christmas period (for sieving). We leave as a subject of further study the trade-off between polynomial search time and the corresponding saving in sieving time.

A.3.2 Sieving

Partially for comparison, two sieving methods were used: lattice sieving and line sieving. The line siever fixes a value of y (from $y = 1, 2, \ldots$ up to some bound) and finds values of x for which both $F_1(x,y)$ and $F_2(x,y)$ are smooth. The lattice siever fixes a prime q, called the special-q, which divides $F_1(x,y)$, and finds (x,y) pairs for which both $F_1(x,y)/q$ and $F_2(x,y)$ are smooth. This is carried out for many special-q's. Lattice sieving was introduced by Pollard [57] and the code we used is the implementation described in [29, 20], with some additions to handle skew sieving regions efficiently.

For the lattice sieving, a rational factor base of 250 000 elements (the primes $\leq 3\,497\,867$) and an algebraic factor base of 800 000 elements (ideals of norm $\leq 12\,174\,433$) were chosen. For the line sieving, larger factor base bounds were chosen, namely: a rational factor base consisting of the primes $< 8\,000\,000$ and an algebraic factor base with the primes $< 16\,777\,216 = 2^{24}$. For both sieves the large prime bounds were 500 000 000 for the rational primes and 1 000 000 000 for the algebraic primes. The lattice siever allowed two large primes on each side, in addition to the special-q input. The line siever allowed three large primes on the algebraic side (this was *two* for RSA-130) and two large primes on the rational side.

The special-q's in the lattice siever were taken from selected parts of the interval $[12\,175\,000, 91\,000\,000]$ and a total of 2 361 390 special-q's were handled. Lattice sieving ranged over a rectangle of 8192 by 4000 points per special-q, i.e., a total of about $7.7 \cdot 10^{13}$ points. Averaged over all the workstations and PCs on which the lattice siever was run, about 52 seconds were needed to handle one special-q and about 16 relations were found per special-q. So on average the lattice siever needed 3.25 CPU seconds to generate one relation.

Line sieving ranged over most of $|x| < 900\,000\,000$ and $1 \leq y \leq 70\,000$, about $1.2 \cdot 10^{15}$ points. It would have been better to reduce the bound on x and raise the bound on y, in accordance with skewness 4000, but we overestimated the amount of line sieving needed. 30% of the relations found with the line-siever had three large primes. Averaged over all the workstations and PCs on which the line siever was run, it needed 5.1 CPU seconds to generate one relation.

A fair comparison of the performances of the lattice and the line siever is difficult for the following reasons: memory requirements of the two sievers are different; the efficiency of both sievers decreases – but probably not with the same "speed" – as the sieving time increases; the codes which we used for lattice and line sieving were optimised by different persons (Arjen Lenstra, resp. Peter Montgomery).

A total of 68 500 867 relations were generated, 56% of them with lattice sieving (indicated below by "LA"), 44% with line sieving (indicated by "LI").

Sieving was done at five different locations with the following contributions:

36.8 %	Peter L. Montgomery, Stefania Cavallar, Herman J.J. te Riele, Walter M. Lioen (LI, LA at CWI, Amsterdam, The Netherlands)
28.8 %	Paul C. Leyland (LA at Microsoft Research Ltd, Cambridge, UK)
26.6 %	Bruce Dodson (LI, LA at Lehigh University, Bethlehem, PA, USA)
5.4 %	Paul Zimmermann (LA at Médicis Center, Palaiseau, France)
2.5 %	Arjen K. Lenstra (LA at Citibank, Parsippany, NJ, USA, and at the University of Sydney, Australia)

Sieving started the day before Christmas 1998 and was completed one month later. Sieving was done on about 125 SGI and Sun workstations running at 175 MHz on average, and on about 60 PCs running at 300 MHz on average. The total amount of CPU time spent on sieving was 8.9 CPU-years. We estimate this to be equivalent to 2000 MIPS years. For comparison, RSA-130 took about 1000 MIPS years. Practical experience we collected with factoring large RSA–numbers tells us that with a careful tuning of the parameters the sieving times may be reduced now to 1000 resp. 500 MIPS years. The relations were collected at CWI and required 3.7 Gbytes of disk storage.

A.3.3 Filtering and finding dependencies

The filtering of the data and the building of the matrix were carried out at CWI and took one calendar week.

Filtering.

Not all the sieved relations were used for filtering since we had to start the huge job for finding dependencies at a convenient moment. We actually used 65.7M of the 68.5M relations as filter input.

First, the "raw" data from the different contributing sites were searched through for duplicates. This single-contributor cleaning removed 1.4M duplicates. Next, we collected all the relations and eliminated duplicates again. This time, 9.2M duplicates were found. The 1.4M + 9.2M duplicates came from machine and human error (e.g., the resumption of early aborted jobs resp. duplicate jobs), from the simultaneous use of the lattice and the line siever, and from the line siever and the lattice siever themselves.

In the filter steps which we describe next, we only considered prime ideals with norm larger than 10 million; in the sequel, we shall refer to these ideals as the *large prime ideals*. In the remaining 55.1M relations we counted 54.1M large prime ideals. We added 0.1M free relations (cf. [28, Section 4, pp. 234–235]). Taking into account another 1.3M prime ideals with norm *below* 10 million, it seemed that we did not have enough relations at this point. However, after we removed 28.5M so-called *singletons* (i.e., relations which contain a large prime ideal that does *not* appear in any other relation) we were left with 26.7M relations having 21.5M large prime ideals. So now we had more than enough relations compared with the total number of prime ideals. We deleted another

17.6M relations which were heuristically judged the least useful,[4] or which became singletons after we had removed some other relations. We were finally left with 9.2M relations containing 7.8M large prime ideals. After this, relations with large prime ideals occurring twice were merged (6.0M relations left) and, finally, those occurring three times were merged (4.7M relations left).

Finding dependencies.

The resulting matrix had 4 671 181 rows and 4 704 451 columns, and weight 151 141 999 (32.36 nonzeros per row). With the help of Peter Montgomery's Cray implementation of the block Lanczos algorithm (cf. [49]) it took almost 100 CPU-hours and 810 Mbytes of central memory on the Cray C916 at the SARA Amsterdam Academic Computer Center to find 64 dependencies among the rows of this matrix. Calendar time for this job was five days.

A.3.4 The square root step

During February 1–2, 1999, four square root (cf. [48]) jobs were started in parallel on four different 250 MHz processors of CWI's SGI Origin 2000, each handling one dependency. Each had about 5 million (not necessarily distinct) $a-b\alpha$ terms in the product. After 14.2 CPU-hours, one of the four jobs stopped, giving the two prime factors of RSA-140. Two others also expired with the two prime factors after 19 CPU-hours (due to different input parameter choices). One of the four jobs expired with the trivial factors.

We found that the 140-digit number

$$\begin{aligned}\text{RSA-140} \;=\; & 2129024631\,8258757547\,4978820162\,7151749780\\ & 6703963277\,2162782333\,8321538194\,9984056495\,9113665738\\ & 5302191831\,6783107387\,9953172308\,8956923087\,3441936471\end{aligned}$$

can be written as the product of the two 70-digit primes:

$$\begin{aligned}\mathrm{p} \;=\; & 3398717423\,0284385545\\ & 3012362761\,3875835633\,9864959695\,9742349092\,9302771479\end{aligned}$$

and

$$\begin{aligned}\mathrm{q} \;=\; & 6264200187\,4012850961\\ & 5165494826\,4442219302\,0371786235\,0901911166\,0653946049.\end{aligned}$$

Primality of the factors was proved with the help of two different primality proving codes [9, 19]. The factorisations of $p \pm 1$ and $q \pm 1$ are given by

$$\begin{aligned}p-1 &= 2 \cdot 7 \cdot 7649 \cdot 435653 \cdot 396004811 \cdot p_{51},\\ p+1 &= 2^3 \cdot 3^2 \cdot 5 \cdot 13 \cdot 8429851 \cdot 33996935324034876299 \cdot p_{40},\\ q-1 &= 2^6 \cdot 61 \cdot 135613 \cdot 3159671789 \cdot p_{52},\\ q+1 &= 2 \cdot 3 \cdot 5^2 \cdot 389 \cdot 6781 \cdot 982954918150967 \cdot p_{47}\end{aligned}$$

[4]The criterion used for this filter step is described in Chapter 3.

where p_x denotes the largest factor, having x digits.

Acknowledgements.

Acknowledgements are due to the Dutch National Computing Facilities Foundation (NCF) for the use of the Cray C916 supercomputer at SARA, and to (in alphabetical order) CWI, Lehigh University, the Magma Group of John Cannon at the University of Sydney, the Médicis Center at École Polytechnique (Palaiseau, France), and Microsoft Research Ltd (Cambridge, UK), for the use of their computing resources.

Appendix B

Factorisation of a 512-bit RSA Modulus

Stefania Cavallar, CWI, Amsterdam, The Netherlands,
Bruce Dodson, Lehigh University, Bethlehem, PA, USA,
Arjen K. Lenstra, Citibank, Mendham, NJ, USA,
Walter Lioen, CWI, Amsterdam, The Netherlands,
Peter L. Montgomery, Microsoft Research, USA and CWI, Amsterdam, The Netherlands,
Brian Murphy, The Australian National University, Canberra, Australia,
Herman te Riele, CWI, Amsterdam, The Netherlands,
Karen Aardal, Utrecht University, Utrecht, The Netherlands,
Jeff Gilchrist, Entrust Technologies Ltd., Ottawa, ON, Canada,
Gérard Guillerm, École Polytechnique, Palaiseau, France,
Paul Leyland, Microsoft Research Ltd., Cambridge, UK,
Joël Marchand, École Polytechnique/CNRS, Palaiseau, France,
François Morain, École Polytechnique, Palaiseau, France,
Alec Muffett, Sun Microsystems, Camberley, UK,
Chris Putnam and Craig Putnam, Hudson, NH, USA,
Paul Zimmermann, Inria Lorraine and Loria, Nancy, France.

Abstract

This paper reports on the factorisation of the 512-bit number RSA-155 by the Number Field Sieve factoring method (NFS) and discusses the implications for RSA.

This chapter is a slight revision of the article "Factorization of a 512-bit RSA modulus" which appeared in the proceedings of the EUROCRYPT 2000 conference [17].

B.1 Introduction

On August 22, 1999, we completed the factorisation of the 512-bit 155-digit number RSA-155 by NFS. The number RSA-155 was taken from the RSA Challenge list [65] as a representative 512-bit RSA modulus. Our result is a new record for factoring general integers.[2] Because 512-bit RSA keys are frequently used for the protection of electronic commerce—at least outside the USA—this factorisation represents a breakthrough in research on RSA–based systems.

The previous record, factoring the 140-digit number RSA-140 (Appendix A), was established on February 2, 1999, also with the help of NFS, by a subset of the team which factored RSA-155. The amount of computing time spent on RSA-155 was about 8400 MIPS years,[3] roughly four times that needed for RSA-140; this is about half of what could be expected from a straightforward extrapolation of the computing time spent on factoring RSA-140 and about a quarter of what would be expected from a straightforward extrapolation of the computing time spent on RSA-130 [20]. The speed-up is due to a new polynomial selection method for NFS of Murphy and Montgomery which was applied for the first time to RSA-140 and now, with improvements, to RSA-155.

Section B.2 discusses the implications of this project for the practical use of RSA–based cryptosystems. Section B.3 has the details of our computations which resulted in the factorisation of RSA-155.

B.2 Implications for the practice of RSA

RSA is widely used today [66]. The best size for an RSA key depends on the security needs of the user and on how long his/her information needs to be protected.

The amount of CPU time spent to factor RSA-155 was about 8400 MIPS years, which is about four times that used for the factorisation of RSA-140. On the basis of the heuristic complexity formula [13] for factoring large N by NFS:

$$\exp\Big((1.923 + o(1))\,(\log N)^{1/3}(\log\log N)^{2/3}\Big), \tag{B.1}$$

one would expect an increase in the computing time by a factor of about seven.[4] This speed-up has been made possible by algorithmic improvements, mainly in

[2]The record as of March 2002 is the factorisation of a 158-digit general number [6] in January 2002.

[3]One *MIPS year* is the equivalent of a computation during one full year at a sustained speed of one **M**illion **I**nstructions **P**er **S**econd.

[4]By "computing time" we mean the *sieve* time, which dominates the total amount of CPU time for NFS. However, there is a trade-off between polynomial search time and sieve time which indicates that a non-trivial part of the total amount of computing time should be spent to the polynomial search time in order to minimise the sieve time. See Subsection *Polynomial Search Time vs. Sieving Time* in Section B.3.1. When we use (B.1) for predicting CPU times, we neglect the $o(1)$–term, which, in fact, is proportional to $1/\log(N)$. All logarithms have base e.

the polynomial generation step [50, 53, 54], and to a lesser extent in the filter step of NFS (see Chapter 3).

The complete project to factor RSA-155 took seven calendar months. The polynomial generation step took about one month on several fast workstations. The most time-consuming step, the sieving, was done on about 300 fast PCs and workstations spread over twelve "sites" in six countries. This step took 3.7 calendar months, in which, summed over all these 300 computers, a total of 35.7 years of CPU-time was consumed. Filtering the relations and building and reducing the matrix corresponding to these relations took one calendar month and was carried out on an SGI Origin 2000 computer. The block Lanczos step to find dependencies in this matrix took about ten calendar days on one CPU of a Cray C916 supercomputer. The final square root step took about two days calendar time on an SGI Origin 2000 computer.

Based on our experience with factoring large numbers we estimate that within three years the algorithmic and computer technology which we used to factor RSA-155 will be widespread, at least in the scientific world, so that by then 512-bit RSA keys will certainly not be safe any more. This makes these keys useless for authentication or for the protection of data required to be secure for a period longer than a few days.

512-bit RSA keys protect 95% of today's E-commerce on the Internet [68]—at least outside the USA—and are used in SSL (Secure Socket Layer) handshake protocols. Underlying this undesirable situation are the old export restrictions imposed by the USA government on products and applications using "strong" cryptography like RSA. However, on January 12, 2000, the U.S. Department of Commerce Bureau of Export Administration (BXA) issued new encryption export regulations which allow U.S. companies to use larger than 512-bit keys in RSA–based products [14]. As a result, one may replace 512-bit keys by 768-bit or even 1024-bit keys thus creating much more favourable conditions for secure Internet communication.

In order to attempt an extrapolation, we give a table of factoring records starting with the landmark factorisation in 1970 by Morrison and Brillhart of $F_7 = 2^{128} + 1$ with help of the then new Continued Fraction (CF) method. This table includes the complete list of factored RSA–numbers, although RSA-100 and RSA-110 were not absolute records at the time they were factored. Notice that RSA-150 is still open. Some details on recent factoring records are given in Appendix B.3.4 to this paper.

Based on this table and on the factoring algorithms which we currently know, we anticipate that within ten years from now 768-bit (232-digit) RSA keys will become unsafe.

Let D be the number of decimal digits in the largest "general" number factored by a given date. From the complexity formula for NFS (B.1), assuming Moore's law (computing power doubles every 18 months), Brent [10] expects $D^{1/3}$ to be roughly a linear function of the calendar year Y. From the data in

# decimals	date or year	algorithm	effort (MIPS years)	reference
39	Sep 1970	CF		$F_7 = 2^{2^7} + 1$ [51, 52]
50	1983	CF		[11, pp. xliv–xlv]
55–71	1983–1984	QS		[22, Table I on p. 189]
45–81	1986	QS		[70, p. 336]
78–90	1987–1988	QS		[69]
87–92	1988	QS		[63, Table 3 on p. 274]
93–102	1989	QS		[41]
107–116	1990	QS	275 for C116	[42]
100	Apr 1991	QS	7	RSA-100 [65]
110	Apr 1992	QS	75	RSA-110 [25]
120	Jun 1993	QS	835	RSA-120 [23]
129	Apr 1994	QS	5000	RSA-129 [3]
130	Apr 1996	NFS	1000	RSA-130 [20]
140	Feb 1999	NFS	2000	RSA-140 Appendix A
155	Aug 1999	NFS	8400	RSA-155 this paper

Table B.1: Factoring records since 1970

Table B.1 he derives the linear formula

$$Y = 13.24D^{1/3} + 1928.6.$$

According to this formula, a general 768-bit number ($D = 231$) will be factored by the year 2010, and a general 1024-bit number ($D = 309$) by the year 2018.

Directions for selecting cryptographic key sizes now and in the coming years are given in [43].

The vulnerability of a 512-bit RSA modulus was predicted long ago. A 1991 report [7, p. 81] recommends:

> For the most applications a modulus size of 1024 bit for RSA should achieve a sufficient level of security for "tactical" secrets for the next ten years. This is for long-term secrecy purposes, for short-term authenticity purposes 512 bit might suffice in this century.

B.3 Factoring RSA-155

We assume that the reader is familiar with NFS [39], but for convenience we briefly describe the method here. Let N be the number we wish to factor, known to be composite. There are four main steps in NFS: polynomial selection, sieving, linear algebra, and square root.

The *polynomial selection step* selects two irreducible polynomials $f_1(x)$ and $f_2(x)$ with a common root m mod N. The polynomials have as many smooth values as practically possible over a given factor base.

The *sieve step* (which is by far the most time-consuming step of NFS), finds pairs (a, b) with $\gcd(a, b) = 1$ such that both

$$b^{\deg(f_1)} f_1(a/b) \text{ and } b^{\deg(f_2)} f_2(a/b)$$

are smooth over given factor bases, i.e., factor completely over the factor bases. Such a pair (a, b) is called a *relation.* The purpose of this step is to collect so many relations that several subsets S of them can be found with the property that a product taken over S yields an expression of the form

$$X^2 \equiv Y^2 \pmod{N}. \tag{B.2}$$

For approximately half of these subsets, computing $\gcd(X - Y, N)$ yields a non-trivial factor of N (if N has exactly two distinct factors).

The *linear algebra step* first filters the relations found during sieving, with the purpose of eliminating duplicate relations and relations containing a prime or prime ideal which does not occur elsewhere. In addition, certain relations are *merged* with the purpose of eliminating primes and prime ideals which occur exactly k times in k different relations, for $k = 2, \ldots, 8$. These merges result in so-called relation–sets, defined in Section B.3.3, which form the columns of a very large sparse matrix over $\mathcal{F}_2$. With help of an iterative block Lanczos algorithm a few dependencies are found in this matrix: this is the most time– and space–consuming part of the linear algebra step.

The *square root step* computes the square root of an algebraic number of the form

$$\prod_{(a,b)\in S} (a - b\alpha),$$

where α is a root of one of the polynomials $f_1(x), f_2(x)$, and where for RSA-155 the numbers a, b and the cardinality of the set S can all be expected to be many millions. All $a - b\alpha$'s have smooth norms. With the mapping $\alpha \mapsto m \bmod N$, this leads to a congruence of the form (B.2).

In the next four subsections, we describe these four steps, as carried out for the factorisation of RSA-155.

B.3.1 Polynomial selection

This section has three parts. The first two parts are aimed at recalling the main details of the polynomial selection procedure, and describing the particular polynomials used for the RSA-155 factorisation.

Relatively speaking, our selection for RSA-155 is approximately 1.7 times better than our selection for RSA-140. We made better use of our procedure for RSA-155 than we did for RSA-140, in short by searching longer. This poses a new question for NFS factorisations—what is the optimal trade-off between increased polynomial search time and the corresponding saving in sieve time? The third part of this section gives preliminary consideration to this question as it applies to RSA-155.

The Procedure

Our polynomial selection procedure is outlined in Appendix A. Here we merely restate the details. Recall that we generate two polynomials f_1 and f_2, using a base-m method. The degree d of f_1 is fixed in advance (for RSA-155 we take $d = 5$). Given a potential a_5, we choose an integer $m \approx (N/a_d)^{1/d}$. The polynomial

$$f_1(x) = a_d x^d + a_{d-1}x^{d-1} + \ldots + a_0 \tag{B.3}$$

descends from the base-m representation of N, initially adjusted so that $|a_i| \leq m/2$ for $0 \leq i \leq d-1$.

Sieving occurs over the homogeneous polynomials $F_1(x, y) = y^d f_1(x/y)$ and $F_2(x, y) = x - my$. The aim for polynomial selection is to choose f_1 and m such that the values $F_1(a, b)$ and $F_2(a, b)$ are simultaneously smooth at many coprime integer pairs (a, b) in the sieving region. That is, we seek F_1, F_2 with good *yield*. Since F_2 is linear, we concentrate on the choice of F_1.

There are two factors which influence the yield of F_1, *size* and *root properties*, so we seek F_1 with a good *combination* of size and root properties. By size we refer to the magnitude of the values taken by F_1. By root properties we refer to the extent to which the distribution of the roots of F_1 modulo small p^n, for p prime and $n \geq 1$, affects the likelihood of F_1 values being smooth. In short, if F_1 has many roots modulo small p^n, the values taken by F_1 "behave" as if they are much smaller than they actually are. That is, on average, the likelihood of F_1-values being smooth is increased.

Our search is a *two stage* process. In the first stage we generate a large sample of good polynomials (polynomials with good combinations of size and root properties). In the second stage we identify *without* sieving, the best polynomials in the sample. We concentrate on *skewed* polynomials, that is, polynomials $f_1(x) = a_5x^5 + \ldots + a_0$ whose first few coefficients (a_5, a_4 and a_3) are small compared to m, and whose last few coefficients (a_2, a_1 and a_0) may be large compared to m. Usually $|a_5| < |a_4| < \cdots < |a_0|$. To compensate for the last few coefficients being large, we sieve over a skewed region, i.e., a region that is much longer in x than in y. We take the region to be a rectangle whose width-to-height ratio is s.

The first stage of the process, generating a sample of polynomials with good yield, has the following main steps ($d = 5$):

- Guess leading coefficient a_d, usually with several small prime divisors (for projective roots).

- Determine initial m from $a_d m^d \approx N$. If the approximation

$$(N - a_d m^d)/m^{d-1}$$

  to a_{d-1} is not close to an integer, try another a_d. Otherwise use (B.3) to determine a starting f_1.

- Try to replace the initial f_1 by a smaller one. This numerical optimisation step replaces $f_1(x)$ by
$$f_1(x+k) + (cx+d)\cdot(x+k-m)$$
and m by $m-k$, sieving over a region with skewness s. It adjusts four real parameters c, d, k, s, rounding the optimal values (except s) to integers.

- Make adjustments to f_1 which cause it to have exceptionally good root properties, *without* destroying the qualities inherited from above. The main adjustment is to consider integer pairs j_1, j_0 (with j_1 and j_0 small compared to a_2 and a_1 respectively) for which the polynomial
$$f_1(x) + (j_1x - j_0)\cdot(x-m)$$
has exceptionally good root properties modulo many small p^n. Such pairs j_1, j_0 are identified using a sieve-like procedure. For each promising (j_1, j_0) pair, we revise the translation k and skewness s by repeating the numerical optimisation on these values alone.

In the second stage of the process we rate, without sieving, the yields of the polynomial pairs F_1, F_2 produced from the first stage. We use a parameter which quantifies the effect of the root properties of each polynomial. We factor this parameter into estimates of smoothness probabilities for F_1 and F_2 across a region of skewness s.

At the conclusion of these two stages we perform short sieving experiments on the top-ranked candidates.

Results

Four of us spent about 100 MIPS years on finding good polynomials for RSA-155. The following pair, found by Dodson, was used to factor RSA-155:

$$\begin{array}{rcrl}
F_1(x,y) & = & 11\,93771\,38320 & x^5 \\
 & & -80\,16893\,72849\,97582 & x^4y \\
 & & -66269\,85223\,41185\,74445 & x^3y^2 \\
 & & +1\,18168\,48430\,07952\,18803\,56852 & x^2y^3 \\
 & & +745\,96615\,80071\,78644\,39197\,43056 & x\ y^4 \\
 & & -40\,67984\,35423\,62159\,36191\,37084\,05064 & \quad y^5 \\
 & & & \\
F_2(x,y) & = & x - 3912\,30797\,21168\,00077\,13134\,49081 & y
\end{array}$$

with $s \approx 10800$.

For the purpose of comparison, we give statistics for the above pair similar to those we gave for the RSA-140 polynomials in Appendix A. Denote by a_{max} the largest $|a_i|$ for $i = 0, \ldots, d$. The un-skewed analogue, $F_1(104x, y/104)$, of F_1 has $a_{max} \approx 1.1 \cdot 10^{23}$, compared to the typical case for RSA-155 of $a_{max} \approx 2.4 \cdot 10^{25}$.

The un-skewed analogue of F_2 has $a_{max} \approx 3.8 \cdot 10^{26}$. Hence, F_1 values have shrunk by approximately a factor of 215, whilst F_2 values have grown by a factor of approximately 16. F_1 has real roots x/y near -11976, -2225, 1584, 12012 and 672167.

With respect to the root properties of F_1 we have $a_5 = 2^4 \cdot 3^2 \cdot 5 \cdot 11^2 \cdot 19 \cdot 41 \cdot 1759$. Also, $F_1(x, y)$ has 20 roots x/y modulo the six primes from 3 to 17 and an additional 33 roots modulo the 18 primes from 19 to 97. As a result of its root properties, F_1-values have smoothness probabilities similar to those of random integers which are smaller by a factor of about 800.

Polynomial Search Time vs. Sieving Time

The yield of our two RSA-155 polynomials is approximately 13.5 times that of a skewed pair of average yield for RSA-155 (about half of which comes from root properties and the other half from size). The corresponding figure for the RSA-140 pair is approximately 8 (about a factor of four of which was due to root properties and the remaining factor of 2 to size). From this we deduce that, relatively speaking, our RSA-155 selection is approximately 1.7 times "better" than our RSA-140 selection.

Note that this is consistent with the observed differences in sieve time. As noted above, straightforward extrapolation of the NFS asymptotic run-time estimate (B.1) suggests that sieving for RSA-155 should have taken approximately 7 times as long as RSA-140. The actual figure is approximately 4. The difference can be approximately reconciled by the fact that the RSA-155 polynomial pair is, relatively, about 1.7 times "better" than the RSA-140 pair.

Another relevant comparison is to the RSA-130 factorisation. RSA-130 of course was factored *without* our improved polynomial selection methods. The polynomial pair used for RSA-130 has a yield approximately 3.2 times that of a random (un-skewed) selection or RSA-130. Extrapolation of the asymptotic NFS run-time estimate suggests that RSA-140 should have taken about 4 times as long as RSA-130, whereas the accepted difference is a factor of about 2. The difference is close to being reconciled by the RSA-140 polynomial selection being approximately 2.5 times better than the RSA-130 selection. Finally, to characterise the overall improvement accounted for by our techniques, we note that the RSA-155 selection is approximately 4.2 times better (relatively) than the RSA-130 selection.

Since the root properties of the non-linear polynomials for RSA-140 and RSA-155 are similar, most of the difference between them comes about because the RSA-155 selection is relatively "smaller" than the RSA-140 selection. This in turns comes about because we conducted a longer search for RSA-155 than we did for the RSA-140 search, so it was more likely that we would find good size and good root properties coinciding in the same polynomials. In fact, we spent approximately 100 MIPS years on the RSA-155 search, compared to 60 MIPS years for RSA-140.

Continuing to search for polynomials is worthwhile only as long as the saving in sieve time exceeds the extra cost of the polynomial search. We have analysed the "goodness" distribution of all polynomials generated during the RSA-155 search. Modulo some crude approximations, the results appear in Table B.2. The table shows the expected benefit obtained from κ times the polynomial search effort we actually invested (100 MY), for some useful κ. The second column gives the change in search time corresponding to the κ-altered search effort. The third column gives the expected change in sieve time, calculated from the change in yield according to our "goodness" distribution. Hence,

κ	change in search time (in MY)	change in sieve time (in MY)
0.2	−80	+260
0.5	−50	+110
1	0	0
2	+100	−110
5	+400	−260
10	+900	−380

Table B.2: Effect of varying the polynomial search time on the sieve time

whilst the absolute benefit may not have been great, it would probably have been worthwhile investing up to about twice the effort than we did for the RSA-155 polynomial search. We conclude that, in the absence of further improvements, it is worthwhile using our method to find polynomials whose yields are approximately 10–15 times better than a random selection.

B.3.2 Sieving

Two sieving methods were used simultaneously: lattice sieving and line sieving. This is probably more efficient than using a single sieve, despite the large percentage of duplicates found (about 14%, see Section B.3.3): both sievers deteriorate as the special q, resp. y (see below) increase, so we exploited the most fertile parts of both. In addition, using two sievers offers more flexibility in terms of memory: lattice sieving is possible on smaller machines; the line siever needs more memory, but discovers each relation only once.

The lattice siever fixes a prime q, called the special q, which divides $F_1(x_0, y_0)$ for some known nonzero pair (x_0, y_0), and finds (x, y) pairs for which both $F_1(x, y)/q$ and $F_2(x, y)$ are smooth. This is carried out for many special q's. Lattice sieving was introduced by Pollard [57] and the code we used is the implementation written by Arjen Lenstra and described in [29, 20], with some additions to handle skewed sieving regions efficiently.

The line siever fixes a value of y (from $y = 1, 2, \ldots$ up to some bound) and finds values of x in a given interval for which both $F_1(x, y)$ and $F_2(x, y)$ are smooth. The line siever code was written by Peter Montgomery, with help from Arjen Lenstra, Russell Ruby, Marije Elkenbracht-Huizing and Stefania Cavallar.

For the lattice sieving, both the rational and the algebraic factor base bounds were chosen to be $2^{24} = 16\,777\,216$. The number of primes was about one million in each factor base. Two large primes were allowed on each side in addition to the special q input. The reason that we used these factor base bounds is that we used the lattice sieving implementation from [29] which does not allow larger factor base bounds. That implementation was written for the factorisation of RSA-130 and was never intended to be used for larger numbers such as RSA-140, let alone RSA-155. We expect that a rewrite of the lattice siever that would allow larger factor base bounds would give a much better lattice sieving performance for RSA-155.

Most of the line sieving was carried out with two large primes on both the rational and the algebraic side. The rational factor base consisted of 2 661 384 primes $< 44\,000\,000$ and the algebraic factor base consisted of 6 304 167 prime ideals of norm $< 110\,000\,000$ (including the seven primes which divide the leading coefficient of $F_1(x, y)$). Some line sieving allowed *three* large primes instead of *two* on the algebraic side. In that case the rational factor base consisted of 539 777 primes $< 8\,000\,000$ and the algebraic factor base of 1 566 598 prime ideals of norm $< 25\,000\,000$ (including the seven primes which divide the leading coefficient of $F_1(x, y)$).

For both sievers the large prime bound 1 000 000 000 was used both for the rational and for the algebraic primes.

The lattice siever was run for most special q's in the interval $[2^{24}, 3.08 \times 10^8]$. Each special q has at least one root r such that $f_1(r) \equiv 0 \bmod q$. For example, the equation $f_1(x) \equiv 0 \bmod q$ has five roots for $q = 83$, namely $x = 8$, 21, 43, 54, 82, but no roots for $q = 31$. The total number of special q–root pairs (q, r) in the interval $[2^{24}, 3.08 \times 10^8]$ equals about 15.7M. Lattice sieving ranged over a rectangle of 8192 by 5000 points per special q–root pair. Taking into account that we did not sieve over points (x, y) where both x and y are even, this gives a total of 4.8×10^{14} sieving points. With lattice sieving a total of 94.8M relations were generated at the expense of 26.6 years of CPU time. Averaged over all the CPUs on which the lattice siever was run, this gives an average of 8.8 CPU seconds per relation.

For the line sieving with two large primes on both sides, sieving ranged over the regions:[5]

$$\begin{aligned} |x| &\le 1\,176\,000\,000, & 1 &\le y \le 25\,000, \\ |x| &\le 1\,680\,000\,000, & 25\,001 &\le y \le 110\,000, \\ |x| &\le 1\,680\,000\,000, & 120\,001 &\le y \le 159\,000, \end{aligned}$$

and for the line sieving with three large primes instead of two on the algebraic side, the sieving range was:

$$|x| \le 1\,680\,000\,000, \quad 110\,001 \le y \le 120\,000.$$

[5]The somewhat weird choice of the line sieving intervals was made because more contributors chose line sieving than originally estimated.

Not counting the points where both x and y are even, this gives a total of 3.82×10^{14} points sieved by the line siever. With line sieving a total of 36.0M relations were generated at the expense of 9.1 years of CPU time. Averaged over all the CPUs on which the line siever was run, it needed 8.0 CPU seconds to generate one relation.

Sieving was done at twelve different locations where a total of 130.8M relations were generated, 94.8M by lattice sieving and 36.0M by line sieving. Each incoming file was checked at the central site for duplicates: this reduced the total number of useful incoming relations to 124.7M. Of these, 88.8M (71%) were found by the lattice siever and 35.9M (29%) by the line siever. The breakdown of the 124.7M relations (in %) among the twelve different sites[6] is given in Table B.3.

%	# CPU days sieved	La(ttice) Li(ne)	Contributor
20.1	3057	La	Alec Muffett
17.5	2092	La, Li	Paul Leyland
14.6	1819	La, Li	Peter L. Montgomery, Stefania Cavallar
13.6	2222	La, Li	Bruce Dodson
13.0	1801	La, Li	François Morain and Gérard Guillerm
6.4	576	La, Li	Joël Marchand
5.0	737	La	Arjen K. Lenstra
4.5	252	Li	Paul Zimmermann
4.0	366	La	Jeff Gilchrist
0.65	62	La	Karen Aardal
0.56	47	La	Chris and Craig Putnam

Table B.3: Breakdown of sieving contributions

Calendar time for the sieving was 3.7 months. Sieving was done on about 160 SGI and Sun workstations (175–400 MHz), on eight R10000 processors (250 MHz), on about 120 Pentium II PCs (300–450 MHz), and on four Digital/Compaq boxes (500 MHz). The total amount of CPU-time spent on sieving was 35.7 CPU years.

We estimate the equivalent number of MIPS years as follows. For each contributor, Table B.4 gives the number of million relations generated (rounded to two decimals), the number of CPU days d_s sieved for this and the estimated average speed s_s, in million instructions per seconds (MIPS), of the processors on which these relations were generated. In the last column we give the corresponding number of MIPS years $d_s s_s/365$. For the time counting on PCs, we notice that on PCs one usually get *real times* which may be higher than the CPU times.

Summarising gives a total of 8360 MIPS years (6570 for lattice and 1790 for line sieving). For comparison, RSA-140 took about 2000 MIPS years and

[6]Lenstra sieved at two sites, viz., Citibank and Univ. of Sydney.

Contributor	# relations	# CPU days sieved	average speed of processors in MIPS	# MIPS years
Muffett, La	27.46M	3057	285	2387
Leyland, La	19.27M	1395	300	1146
Leyland, Li	4.52M	697	300	573
CWI, La	1.60M	167	175	80
CWI, Li, 2LP	15.64M	1160	210	667
CWI, Li, 3LP	1.00M	492	50	67
Dodson, La	10.28M	1631	175	782
Dodson, Li	7.00M	591	175	283
Morain, La	15.83M	1735	210	998
Morain, Li	1.09M	66	210	38
Marchand, La	7.20M	522	210	300
Marchand, Li	1.11M	54	210	31
Lenstra, La	6.48M	737	210	424
Zimmermann, Li	5.64M	252	195	135
Gilchrist, La	5.14M	366	350	361
Aardal, La	0.81M	62	300	51
Putnam, La	0.76M	47	300	39

Table B.4: # MIPS years spent on lattice (La) and line (Li) sieving

RSA-130 about 1000 MIPS years.

A measure of the "quality" of the sieving may be the average number of points sieved to generate one relation. Table B.5 gives this quantity for RSA-140 and for RSA-155, for the lattice siever and for the line siever. This illustrates that the sieving polynomials were better for RSA-155 than for RSA-140, especially for the line sieving. In addition, the increase of the linear factor base bound from 500M for RSA-140 to 1000M for RSA-155 accounts for some of the change in yield. For RSA-155, the factor bases were much bigger for line sieving than for lattice sieving. This explains the increase of efficiency of the line siever compared with the lattice siever from RSA-140 to RSA-155.

	lattice siever	line siever
RSA-140	1.5×10^6	3.0×10^7
RSA-155	5.1×10^6	1.1×10^7

Table B.5: Average number of points sieved per relation

B.3.3 Filtering and finding dependencies

The filtering of the data and the building of the matrix were carried out at CWI and took one calendar month.

Filtering

Here we describe the filter strategy which we used for RSA-155. An essential difference with the filter strategy used for RSA-140 is that we applied k-way merges (defined below) with $2 \leq k \leq 8$ for RSA-155, but only 2- and 3-way merges for RSA-140.

First, we give two definitions. A *relation–set* is one relation, or a collection of two or more relations generated by a merge. A *k-way merge* ($k \geq 2$) is the action of combining k relation–sets with a common prime ideal into $k-1$ relation–sets, with the purpose of eliminating that common prime ideal. This is done such that the weight increase is minimal by means of a *minimum spanning tree* algorithm (see Chapter 3).

Among the 124.7M relations collected from the twelve different sites, 21.3M duplicates were found generated by lattice sieving, as well as 17.9M duplicates caused by the simultaneous use of the lattice and the line siever.

During the first filter round, only prime ideals with norm $>$ 10M were considered. In a later stage of the filtering, this 10M-bound was reduced to 7M, in order to improve the possibilities for merging relations. We added 0.2M *free relations* for prime ideals of norm $>$ 10M (cf. [28, Section 4, pp. 234–235]). From the resulting 85.7M relations, 32.5M singletons were deleted, i.e., those relations with a prime ideal of norm $>$ 10M which does not occur in any other undeleted relation.

We were left with 53.2M relations containing 42.6M different prime ideals of norm $>$ 10M. If we assume that each prime and each prime ideal with norm $<$ 10M occurs at least once, then we needed to reserve at least $(2 - \frac{1}{120})\pi(10^7)$ excess relations for the primes and the prime ideals of norm smaller than 10M, where $\pi(x)$ is the number of primes below x. The factor 2 comes from the *two* polynomials and the correction factor 1/120 takes account of the presence of free relations, where 120 is the order of the Galois group of the algebraic polynomial. With $\pi(10^7) = 664\ 579$ the required excess is about 1.3M relations, whereas we had 53.2M $-$ 42.6M $=$ 10.6M excess relations at our disposal.

In the next *merging* step 33.0M relations were removed which would have formed the heaviest relation–sets when performing 2-way merges, reducing the excess from 10.6M to about 2M relations. So we were still allowed to discard about 2.0M $-$ 1.3M $=$ 0.7M relations. The remaining 20.1M non-free relations[7] having 18.2M prime ideals of norm $>$ 10M were used as input for the merge step which eliminated prime ideals occurring in up to eight different relation–sets. During this step we looked at prime ideals of norm $>$ 7M. Here, our approach differs from what we did for RSA-140, where only primes occurring twice or thrice were eliminated. Applying the new filter strategy to RSA-140 would have resulted in a 30% smaller (3.3M instead of 4.7M columns) but only 20% heavier matrix than the one actually used for the factorisation of RSA-140 and would have saved 27% on the block Lanczos run time. The k ($k \leq 8$)

[7]The 0.1M free relations are not counted in these 20.1M relations because the free relations are generated during each filter run.

relations were combined into the lightest possible $k-1$ relation–sets and the corresponding prime ideal (row in the matrix) was "balanced" (i.e., all entries of the row were made 0). The overall effect was a reduction of the matrix size by one row and one column while increasing the matrix weight when $k > 2$, as described below. We did not perform all possible merges. We limited the program to only do merges which caused a weight increase of at most 7 original relations. The merges were done in ascending order of weight increase.

Since each k-way merge causes an increase of the matrix weight of about $(k-2)$ times the weight of the lightest relation–set, these merges were not always executed for higher values of k. For example, 7- and 8-way merges were not executed if all the relation–sets were already-combined relations. We decided to discard relation–sets which contained more than 9 relations and to stop merging (and discarding) after 670K relations were discarded. At this point we should have slightly more columns than rows and did not want to lose any more columns. The maximum discard threshold was reached during the 10th pass through the 18.6M prime ideals of norm > 7M, when we allowed the maximum weight increase to be about 6 relations. This means that no merges with weight increase of 7 relations were executed. The filter program stopped with 6.7M relation sets.

For more details and experiments with RSA-155 and other numbers, see Chapter 3.

Finding dependencies

From the matrix left after the filter step we omitted the small primes < 40, thus reducing the weight by 15%. The resulting matrix had 6 699 191 rows, 6 711 336 columns, and weight 417 132 631 (62.27 non–zeros per row). With the help of Peter Montgomery's Cray implementation of the block Lanczos algorithm (cf. [49]) it took 224 CPU hours and 2 Gbytes of central memory on the Cray C916 at the SARA Amsterdam Academic Computer Center to find 64 dependencies among the rows of this matrix. Calendar time for this job was 9.5 days.

In order to extract from these 64 dependencies some dependencies for the matrix *including the primes* < 40, quadratic character checks were used as described in [1], [13, §8, §12.7], and [27, last paragraph of Section 3.8 on pp. 30–31]. This yielded a dense 100×64 homogeneous system which was solved by Gaussian elimination. That system turned out to have 14 independent solutions, which represent linear combinations of the original 64 dependencies.

B.3.4 The square root step

On August 20, 1999, four different square root (cf. [48]) jobs were started in parallel on four different 300 MHz processors of an SGI Origin 2000, each handling one dependency. One job found the factorisation after 39.4 CPU-hours, the other three jobs found the trivial factorisation after 38.3, 41.9, and 61.6

CPU-hours (different CPU times are due to the use of different parameters in the four jobs).

We found that the 155-digit number

$$
\begin{aligned}
\text{RSA-155} \; = \; & \qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad\qquad 10941 \\
& 7386415705\,2742180970\,7322040357\,6120037329\,4544920599 \\
& 0913842131\,4763499842\,8893478471\,7997257891\,2673324976 \\
& 2575289978\,1833797076\,5372440271\,4674353159\,3354333897
\end{aligned}
$$

can be written as the product of two 78-digit primes:

$$
\begin{aligned}
\mathrm{p} \; = \; & \qquad\qquad\qquad\qquad 10263959\,2829741105\,7720541965 \\
& 7399167590\,0716567808\,0380668033\,4193352179\,0711307779
\end{aligned}
$$

and

$$
\begin{aligned}
\mathrm{q} \; = \; & \qquad\qquad\qquad\qquad 10660348\,8380168454\,8209272203 \\
& 6001287867\,9207958575\,9892915222\,7060823719\,3062808643.
\end{aligned}
$$

Primality of the factors was proved with the help of two different primality proving codes [9, 19]. The factorisations of $p \pm 1$ and $q \pm 1$ are given by

$$
\begin{aligned}
p-1 &= 2 \cdot 607 \cdot 305999 \cdot p_{69} \\
p+1 &= 2^2 \cdot 3 \cdot 5 \cdot 5253077241827 \cdot p_{63} \\
q-1 &= 2 \cdot 241 \cdot 430028152261281581326171 \cdot p_{51} \\
q+1 &= 2^2 \cdot 3 \cdot 130637011 \cdot 237126941204057 \cdot 10200242155298917871797 \cdot \\
& \quad p_{32}
\end{aligned}
$$

where p_x denotes the largest factor, having x digits.

Acknowledgements.

Acknowledgements are due to the Dutch National Computing Facilities Foundation (NCF) for the use of the Cray C916 supercomputer at SARA, and to (in alphabetical order) the Australian National University (Canberra), Centre Charles Hermite (Nancy, France), Citibank (Parsippany, NJ, USA), CWI (Amsterdam, The Netherlands), École Polytechnique/CNRS (Palaiseau, France), Entrust Technologies Ltd. (Ottawa, Canada), Lehigh University (Bethlehem, PA, USA), the Magma Group of John Cannon at the University of Sydney, the Medicis Center at École Polytechnique (Palaiseau, France), Microsoft Research (Cambridge, UK), the Putnams (Hudson, NH, USA), Sun Microsystems Professional Services (Camberley, UK), and Utrecht University (The Netherlands), for the use of their computing resources.

Appendix to Appendix B. Details of recent absolute and SNFS factoring records

# digits	129	130	140	155
method	QS	GNFS	GNFS	GNFS
code	Gardner	RSA-130	RSA-140	RSA-155
factor date	Apr 2, 1994	Apr 10, 1996	Feb 2, 1999	Aug 22, 1999
size of p, q	64, 65	65, 65	70, 70	78, 78
sieve time (in MIPS years)	5000	1000	2000	8400
total sieve time (in CPU years)	?	?	8.9	35.7
calendar time for sieving (in days)	~270	120	30	110
matrix size	0.6M	3.5M	4.7M	6.7M
row weight	47	40	32	62
Cray CPU hours	n.a.	67	100	224
group	Internet	Internet	CABAL	CABAL

Table B.6: Absolute factoring records

# digits	148[40]	167	180	186	211
code	2,512+	3,349−	12,167+	NEC	10,211−
factor date	Jun 15, 1990	Feb 4, 1997	Sep 3, 1997	Sep 15, 1998	April 8, 1999
size of p, q	49, 99	80, 87	75, 105	71, 73	93, 118
total sieve time (in CPU years)	340[a]	?	1.5	5.1	10.9
calendar time for sieving (in days)	83	?	10	42	64
matrix size	72K	?	1.9M	2.5M	4.8M
row weight	dense	?	29	27	49
Cray CPU hours	3[b]	?	16	25	121
group	Internet	NFSNET	CWI	CWI	CABAL

[a]MIPS years
[b]carried out on a Connection Machine

Table B.7: Special Number Field Sieve factoring records

Bibliography

[1] L. M. Adleman. Factoring numbers using singular integers. In *Proc. 23rd Annual ACM Symp. on Theory of Computing*, pages 64–71, New York, 1991. ACM.

[2] F. Almgren, G. Andersson, T. Granlund, L. Ivansson, and S. Ulfberg. How we cracked the codebook ciphers. http://codebook.org/.

[3] D. Atkins, M. Graff, A. K. Lenstra, and P. C. Leyland. The magic words are squeamish ossifrage. In J. Pieprzyk and R. Safavi-Naini, editors, *Advances in Cryptology - ASIACRYPT '94*, volume 917 of *Lecture Notes in Computer Science*, pages 263–277, Berlin, 1995. Springer.

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

[5] E. Bach and R. Peralta. Asymptotic semismoothness probabilities. *Math. Comp.*, 65(216):1701–1715, 1996.

[6] F. Bahr, J. Franke, and T. Kleinjung. 2.953+ c158, January 2002. http://www.loria.fr/~zimmerma/records/gnfs158.

[7] T. Beth, M. Frisch, and G. J. Simmons, editors. *Public–Key Cryptography: State of the Art and Future Directions — Report on workshop at Oberwolfach, 1991*, volume 578 of *Lecture Notes in Computer Science*. Springer, Berlin, 1992.

[8] H. Boender. Factoring large integers with the quadratic sieve. Dissertation, Rijksuniversiteit Leiden, 1997.

[9] W. Bosma and M.-P. van der Hulst. Primality proving with cyclotomy. Dissertation, Universiteit van Amsterdam, 1990.

[10] R. P. Brent. Some parallel algorithms for integer factorisation. In *Proc. Europar'99 (Toulouse, Sept. 1999)*, volume 1685 of *Lecture Notes in Computer Science*, pages 1–22, Berlin, 1999. Springer.

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

[12] N. G. de Bruijn. On the number of positive integers $\leq x$ and free of prime factors $> y$. *Indag. Math.*, 13:50–60, 1951.

[13] J. P. Buhler, H. W. Lenstra, Jr., and C. Pomerance. Factoring integers with the number field sieve. In Lenstra and Lenstra [39], pages 50–94.

[14] http://www.bxa.doc.gov/encryption/default.htm.

[15] S. Cavallar. Strategies in filtering in the number field sieve. In W. Bosma, editor, *Algorithmic Number Theory - ANTS-IV*, volume 1838 of *Lecture Notes in Computer Science*, pages 209–231, Berlin, 2000. Springer.

[16] S. Cavallar, B. Dodson, A. Lenstra, P. Leyland, W. Lioen, P. L. Montgomery, B. Murphy, H. te Riele, and P. Zimmermann. Factorization of RSA-140 using the number field sieve. In K. Y. Lam, E. Okamoto, and C. Xing, editors, *Advances in Cryptology – ASIACRYPT '99*, volume 1716 of *Lecture Notes in Computer Science*, pages 195–207. Springer, 1999.

[17] S. Cavallar, B. Dodson, A. K. Lenstra, W. Lioen, P. L. Montgomery, B. Murphy, H. te Riele, K. Aardal, J. Gilchrist, G. Guillerm, P. Leyland, J. Marchand, F. Morain, A. Muffett, C. and C. Putnam, and P. Zimmermann. Factorization of a 512-bit RSA modulus. In B. Preneel, editor, *Advances in Cryptology – EUROCRYPT 2000*, volume 1807, pages 1–18. Springer, 2000.

[18] H. Cohen. *A Course in Computational Algebraic Number Theory.* Springer, 1996.

[19] H. Cohen and A. K. Lenstra. Implementation of a new primality test. *Math. Comp.*, 48:103–121, 1987.

[20] J. Cowie, B. Dodson, R. M. Elkenbracht-Huizing, A. K. Lenstra, P. L. Montgomery, and J. Zayer. A world wide number field sieve factoring record: on to 512 bits. In K. Kim and T. Matsumoto, editors, *Advances in Cryptology – ASIACRYPT '96*, volume 1163 of *Lecture Notes in Computer Science*, pages 382–394, Berlin, 1996. Springer.

[21] http://www.crestco.co.uk/.

[22] J. A. Davis, D. B. Holdridge, and G. J. Simmons. Status report on factoring (at the Sandia National Laboratories). In T. Beth, N. Cot, and I. Ingemarsson, editors, *Advances in Cryptology – EUROCRYPT '84*, volume 209 of *Lecture Notes in Computer Science*, pages 183–215, Berlin, 1985. Springer.

[23] T. 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, Berlin, 1994. Springer.

[24] T. F. Denny. Solving large sparse systems of linear equations over finite prime fields. Transparencies of a lecture of the Cryptography Working Group at CWI, May 1995.

[25] B. Dixon and A. K. Lenstra. Factoring using SIMD sieves. In T. Helleseth, editor, *Advances in Cryptology – EUROCRYPT '93*, volume 765 of *Lecture Notes in Computer Science*, pages 28–39, Berlin, 1994. Springer.

[26] B. Dodson and A. K. Lenstra. NFS with four large primes: An explosive experiment. In D. Coppersmith, editor, *Advances in Cryptology – CRYPTO '95*, volume 963 of *Lecture Notes in Computer Science*, pages 372–385, Berlin, 1995. Springer.

[27] M. Elkenbracht-Huizing. Factoring integers with the number field sieve. Dissertation, Rijksuniversiteit Leiden, 1997.

[28] R.-M. Elkenbracht-Huizing. An implementation of the Number Field Sieve. *Experiment. Math.*, 5(3):231–253, 1996.

[29] R. Golliver, A. K. Lenstra, and K. S. McCurley. Lattice sieving and trial division. In L. M. Adleman and M.-D. Huang, editors, *Algorithmic Number Theory, ANTS-I*, volume 877 of *Lecture Notes in Computer Science*, pages 18–27, Berlin, 1994. Springer.

[30] R. L. Graham and P. Hell. On the history of the minimum spanning tree problem. *Annals of the History of Computing*, 7(1):43–57, 1985.

[31] S. Hunter and J. Sorenson. Approximating the number of integers free of large prime factors. *Math. Comp.*, 66(220):1729–1741, 1997.

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

[33] D. E. Knuth. *The Stanford GraphBase: A Platform for Combinatorial computing.* Addison-Wesley, 1993.

[34] D. E. Knuth. *Seminumerical Algorithms*, volume 2 of *The Art of Computer Programming.* Addison-Wesley, third edition, 1998.

[35] D. E. Knuth. *Sorting and Searching*, volume 3 of *The Art of Computer Programming.* Addison-Wesley, second edition, 1998.

[36] 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 – CRYPTO '90*, volume 537 of *Lecture Notes in Computer Science*, pages 109–133, Berlin, 1991. Springer.

[37] R. Lambert. *Computational aspects of discrete logarithms.* PhD thesis, University of Waterloo, 1996.

[38] S. Lang. *Algebraic Number Theory.* Springer, 1986.

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

[40] A. K. Lenstra, H. W. Lenstra, Jr., M. S. Manasse, and J. M. Pollard. The factorization of the Ninth Fermat number. *Math. Comp.*, 61(203):319–349, 1993.

[41] A. K. Lenstra and M. S. Manasse. Factoring by electronic mail. In J.-J. Quisquater and J. Vandewalle, editors, *Advances in Cryptology – EUROCRYPT '89*, volume 434 of *Lecture Notes in Computer Science*, pages 355–371, Berlin, 1990. Springer.

[42] A. K. Lenstra and M. S. Manasse. Factoring with two large primes. In I. B. Damgård, editor, *Advances in Cryptology – EUROCRYPT '90*, volume 473 of *Lecture Notes in Computer Science*, pages 72–82, Berlin, 1991. Springer.

[43] A. K. Lenstra and E. R. Verheul. Selecting cryptographic key sizes. In H. Imai and Y. Zheng, editors, *Public Key Cryptography*, volume 1751 of *Lecture Notes in Computer Science*, pages 446–465. Springer, 2000.

[44] The magma computational algebra system. http://www.maths.usyd.edu.au:8000/u/magma/.

[45] G. Marsaglia, A. Zaman, and J. C. W. Marsaglia. Numerical solution of some classical differential-difference equations. *Math. Comp.*, 53(187):191–201, 1989.

[46] P. L. Montgomery. Speeding the Pollard and elliptic curve methods of factorization. *Math. Comp.*, 48(177):243–264, 1987.

[47] P. L. Montgomery. Comments in rootfinder program, 1992.

[48] P. L. Montgomery. Square roots of products of algebraic numbers. In W. Gautschi, editor, *Mathematics of Computation 1943–1993: a Half-Century of Computational Mathematics*, volume 48 of *Proceedings of Symposia in Applied Mathematics*, pages 567–571. AMS, 1994.

[49] 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, Berlin, 1995. Springer.

[50] P. L. Montgomery and B. Murphy. Improved polynomial selection for the number field sieve. In *Proc. of the Conference on the Mathematics of Public-Key Cryptography*, The Fields Institute, Toronto, 1999.

[51] M. A. Morrison and J. Brillhart. The factorization of F_7. *Bull. Amer. Math. Soc.*, 77(2), 1971.

[52] M. A. Morrison and J. Brillhart. A method of factoring and the factorization of F_7. *Math. Comp.*, 29:183–205, 1975.

[53] B. Murphy. Modelling the yield of number field sieve polynomials. In J. Buhler, editor, *Algorithmic Number Theory, ANTS-III*, volume 1423 of *Lecture Notes in Computer Science*, pages 137–151. Springer, 1998.

[54] B. A. Murphy. *Polynomial Selection for the Number Field Sieve Integer Factorisation Algorithm.* PhD thesis, The Australian National University, 1999.

[55] P. Nguyen. A Montgomery-like square root for the number field sieve. In J. P. Buhler, editor, *Algorithmic Number Theory - ANTS-III*, volume 1423 of *Lecture Notes in Computer Science*, pages 151–168. Springer, 1998.

[56] J. M. Pollard. Factoring with cubic integers. In Lenstra and Lenstra [39], pages 4–10.

[57] J. M. Pollard. The lattice sieve. In Lenstra and Lenstra [39], pages 43–49.

[58] C. Pomerance. The quadratic sieve factoring algorithm. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Advances in Cryptology – EUROCRYPT '84*, volume 209 of *Lecture Notes in Computer Science*, pages 169–182, New York, 1985. Springer.

[59] C. Pomerance and J. W. Smith. Reduction of huge, sparse matrices over finite fields via created catastrophes. *Experiment. Math.*, 1(2):89–94, 1992.

[60] H.J.J. te Riele. 186-digit SNFS factorization: On to factoring a 512 bits RSA key with GNFS. Available from ftp://ftp.cwi.nl/pub/herman/NFSrecords/SNFS-186, September 1998.

[61] H.J.J. te Riele. 211-digit SNFS factorization. Available from ftp://ftp.cwi.nl/pub/herman/NFSrecords/SNFS-211, April 1999.

[62] H.J.J. te Riele. 233-digit SNFS factorization. Available from ftp://ftp.cwi.nl/pub/herman/SNFSrecords/SNFS-233, November 2000.

[63] H. te Riele, W. Lioen, and D. Winter. Factoring with the quadratic sieve on large vector computers. *J. Comp. Appl. Math.*, 27:267–278, 1989.

[64] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Comm. ACM*, 21:120–126, 1978.

[65] RSA Challenge Administrator. http://www.rsa.com/rsalabs/challenges/factoring.html.

[66] RSA Laboratories' Frequently Asked Questions about Today's Cryptography 4.1. http://www.rsasecurity.com/rsalabs/faq/3-1-9.html.

[67] L. Schoenfeld. Sharper bounds for the Chebyshev functions $\theta(x)$ and $\psi(x)$. ii. *Math. Comp.*, 30(134):337–360, 1976.

[68] A. Shamir. Factoring large numbers with the TWINKLE device. In C. K. Koc and C. Paar, editors, *Cryptographic Hardware and Embedded Systems (CHES)*, volume 1717 of *Lecture Notes in Computer Science*. Springer, 1999.

[69] R. D. Silverman. Private communication.

[70] R. D. Silverman. The multiple polynomial quadratic sieve. *Math. Comp.*, 48(177):329–339, 1987.

[71] A. M. Vershik. The asymptotic distribution of factorizations of natural numbers into prime divisors. *Soviet Math. Dokl.*, 34(1):57–61, 1987.

Samenvatting

De Number Field Sieve is de asymptotisch snelste methode om grote gehele getallen in priemfactoren te ontbinden. In dit proefschrift gaan we in op twee onderdelen van het algoritme, namelijk het zeven en het filteren.

Hoofdstuk 2 - Zeven met drie grote priemfactoren

In het tweede hoofdstuk is gekeken naar de drie-grote-priemgetallen variant van de zeef. De zeef maakt een groot aantal relaties aan. Dat zijn paren van gehele getallen waarover twee voorafbepaalde polynomen tegelijkertijd een 'gladde' waarde hebben, dat wil zeggen, deze polynoomwaarden hebben alleen 'kleine' priemfactoren. Meestal worden voor de huidige grootte van getallen, behalve factoren in de factorbasis, ten hoogste twee grote priemgetallen voor elk van de twee polynoomwaarden toegestaan. In de drie-grote-priemgetallen variant worden aanvullend voor één van de twee polynomen polynoomwaarden toegestaan die drie grote priemfactoren bevatten.

Om een idee te krijgen over de bruikbaarheid van de drie-grote-priemgetallen variant hebben we geprobeerd het aantal relaties met een gegeven aantal grote priemgetallen te voorspellen, dus zónder te zeven. Een eenvoudige methode om het aantal gladde (in dit geval: zonder priemfactoren boven de grens voor de factorbasis) random getallen te benaderen is met behulp van Dickman's rho-functie. Bach en Peralta hebben dit gegeneraliseerd om het aantal gladde getallen met één extra grote priemfactor te kunnen voorspellen. Lambert keek naar het geval met twee extra grote priemfactoren en wij naar drie of meer extra grote priemgetallen. Door een heuristisch idee van Montgomery kan hierbij worden overgegaan van random getallen naar polynoomwaarden. De rho-functie hebben wij numeriek benaderd door de functie stuksgewijs als Taylorreeks te schrijven en ons te beperken tot een minimum aantal termen in de reeks, afhankelijk van de benodigde nauwkeurigheid. Wij hebben dit op twee manieren gedaan: de manier van Patterson en Rumsey om de reeks aan het rechter intervaleinde te expanderen en de manier van Marsaglia, Zaman en Marsaglia, die vanuit het midden van het interval expandeert. Wij konden beide methoden verbeteren door twee formules voor bepaalde Taylorcoëfficienten te vereenvoudigen.

De benaderingen met de rho-functie geven een indicatie van de verhoudingen van de aantallen relaties met 0, 1, 2, 3 grote priemgetallen. Wij hebben de drie-grote-priemgetallen methode ook geïmplementeerd en enkele getallen daarmee ontbonden. Tot nog toe is er nog geen opmerkelijke verbetering ten opzichte van de twee-grote-priemgetallen methode te meten.

Hoofdstuk 3 - Verbetering van de filterstap

In het derde hoofdstuk behandelen we het 'filteren' van relaties. Het bevat een beschrijving van de veranderingen die we hebben aangebracht in het oorspronkelijke filteralgoritme zoals beschreven in [28]. Het algoritme krijgt de relaties uit de zeef als invoer en levert, afhankelijk van de gekozen parameters, al dan niet gecombineerde relaties als uitvoer. Het uiteindelijke doel is relaties zodanig te combineren dat het product (per polynoom) van hun polynoomwaarden een kwadraat vormt. Het filteralgoritme is een voorbereiding daarop. We kunnen de relaties als vectoren beschouwen met als indices de priemgetallen[8] in de factor basis, de voorkomende grote priemgetallen en twee indices voor het voorteken van de polynoomwaarden. Een element in de vector is 0 als het corresponderende priemgetal in de betreffende relatie tot een even macht voorkomt, anders is het 1. Het voorteken-element is 0 als de polynoomwaarde positief is en 1 als hij negatief is. Het filteralgoritme elimineert dubbele relaties en zogenaamde singletons, dat zijn relaties waar een priemgetal in voorkomt dat elders niet voorkomt. Verder zorgt het zogenaamde 'clique'-algoritme voor het selecteren en verwijderen van verzamelingen (cliques) relaties die te 'zwaar' worden geacht. In een tweede fase van het filteralgoritme worden relaties opgespoord die hetzelfde priemgetal bevatten. Deze worden dan tot groepjes van twee relaties gecombineerd om het priemgetal te neutraliseren (er komt een 0 te staan in het betreffende element van de vector van de gecombineerde relaties). In de meeste gevallen wordt simpelweg een pivotrelatie gekozen, die met elk van de andere relaties wordt gecombineerd. In sommige gevallen is het echter beter (vanwege minder enen in de matrix) om door een 'minimum spanning tree' algoritme de lichtste combinatie uit te laten zoeken. De aan het einde overgebleven relatie-verzamelingen worden de kolommen van de matrix die vervolgens aan het Block Lanczos algoritme wordt overgedragen. In dit hoofdstuk beschrijven we verder experimenten met het vernieuwde filteralgoritme die laten zien dat we daarmee (kleinere en dichtere) matrices kunnen verkrijgen die sneller door het Block Lanczos algoritme kunnen worden verwerkt.

Bijlagen

Als bijlagen zijn twee artikelen opgenomen over de record-ontbindingen van RSA-140 (februari 1999) en van RSA-155 (augustus 1999). De auteur van dit

[8]Eigenlijk priemidealen, maar voor het gemak zullen we het hier alleen over priemgetallen hebben.

proefschrift heeft hieraan meegewerkt, als afgeleide van het in dit proefschrift beschreven onderzoek. De in Hoofdstuk 2 beschreven drie-grote-priemgetallen variant werd bij beide ontbindingen gebruikt, bij RSA-140 meer dan bij RSA-155. De in Hoofdstuk 3 beschreven modificaties van de filter-stap zijn bij de ontbinding van RSA-155 toegepast.

Curriculum Vitae

Stefania Cavallar was born in Meran/Merano, Italy on 23rd January 1971. She graduated from Handelsoberschule "Franz Kafka" in Meran in 1990. In the same year she entered the Università degli Studi di Trento, Italy where she studied mathematics.

From 1994 to 1995 she worked as a part time teacher for physics at Lehrerbildungsanstalt "Josef Ferrari" in Meran. In 1996, she earned the *laurea in matematica* with *110/110 e lode.* From 1996 to 1997 she taught mathematics and physics at Humanistisches Gymnasium "Beda Weber" in Meran.

From 1997 to 2001 she was employed as an *onderzoeker in opleiding* at Centrum voor Wiskunde en Informatica (CWI) in Amsterdam working under the supervision of dr.ir. Herman te Riele and prof.dr. Robert Tijdeman.

Since 2001 she has been working as a *toegevoegd onderzoeker* at Technische Universiteit Eindhoven in the project CyberVote.

Stellingen behorende bij het proefschrift

On the Number Field Sieve Integer Factorisation Algorithm

van

Stefania Cavallar

1. De priemfactoren van $\frac{12^{167}+1}{13}$ zijn

$$\begin{array}{r} 16311 \\ 7845256502\,9206875585\,4365669702\,0247411364\,4621203858 \\ 9316094580\,4553250187\,4726047433\,2647643552\,2680378897 \end{array}$$

en

$$\begin{array}{r} 78853\,9152479959\,9235834738 \\ 7072972515\,8796647538\,8837188632\,6218141334\,7391236469. \end{array}$$

Peter Montgomery, Stefania Cavallar, and Herman te Riele. A new world record for the special number field sieve factoring method. *CWI Quarterly*, 10(2):105–107, 1997.

Dit proefschrift.

2. De 99-cijferige deler

$$\begin{array}{rcl} N & = & 155980561\ 6886387152\ 2823398590\ 3716632660\ 7959612089 \\ & & 5335641907\ 8031649173\ 1672845644\ 9496435148\ 3415681737 \end{array}$$

van $29^{171}+1$ kan ontbonden worden door een combinatie van SNFS en GNFS. Hierbij kan men gebruik maken van de polynomen

$$\begin{array}{rcl} f_1(x) & = & 9x^4 - 27x^2 + 25 \\ f_2(x) & = & 41975116119497006751294 36x^3 \\ & & -6312727184512501697639130x^2 \\ & & -8131636782285916429911022x \\ & & 3970227321732550195181799 \end{array}$$

die de wortel $m = \frac{\sqrt{57}+\sqrt{-3}}{6} \bmod N$ gemeen hebben.

Dit proefschrift.

3. Laat bij het klassieke lijn na lijn zeven het zeefvlak gegeven zijn door $[-A, A) \times [1, B] \quad \bigcap \quad \mathbb{Z} \times \mathbb{N}$. Het is gunstig om $B < A$ te hebben, omdat dan minder vaak een nieuw array geïnitialiseerd hoeft te worden. Daarom is het soms beter om in plaats van $f_i(x)$, $i = 1, 2$ met gemeenschappelijke

wortel m mod N die veel kleiner is dan N de polynomen $f_i(1/x)$, $i = 1, 2$ met gemeenschappelijke wortel m^{-1} mod N te kiezen.

Dit proefschrift.

4. De drie-grote-priemgetallen variant kan zowel op de getallenlichamenzeef (NFS) als op de kwadratische zeef (QS) worden toegepast. Men kan verwachten dat voor de QS het omslagpunt tussen de twee-grote-priemgetallen variant en de drie-grote-priemgetallen variant lager ligt dan bij de NFS.

   Dit proefschrift.

5. Het getallenlichaam $K = \mathbb{Q}(\alpha)$ met $\alpha^3 - \alpha^2 - 10\alpha - 3 = 0$, met $\Delta_K = 3305$, heeft

$$\sup_{\xi \in K} \inf_{\eta \in \mathcal{O}_K} |N_{K/\mathbb{Q}}(\xi - \eta)| = \frac{13}{9}$$

   en is dus niet euclidisch ten opzichte van de absolute waarde van de norm.

   Stefania Cavallar and Franz Lemmermeyer. The euclidean algorithm in cubic number fields. In Győry, Pethő, and Sós, uitgevers, *Number Theory - Eger 1996*, pages 123–146. Gruyter, 1998.

6. Het getallenlichaam uit de vorige stelling is euclidisch ten opzichte van de gewogen norm

$$\mathcal{F}_{\mathfrak{p},c}(\mathfrak{q}) = \begin{cases} N\mathfrak{q} & \text{if } \mathfrak{q} \neq (3, \alpha) \\ c & \text{if } \mathfrak{q} = (3, \alpha) \end{cases}$$

   gedefineerd over de priemidealen in $\mathcal{O}_K$ waarbij N de gebruikelijke norm voor priemidealen in $\mathcal{O}_K$ is en c een willekeurig getal in het interval $(\sqrt{13}, 5)$.

   Stefania Cavallar and Franz Lemmermeyer. Euclidean windows. *The LMS Journal of Computation and Mathematics*, 3:336–355, 2000. http://www.lms.ac.uk/jcm/3/lms2000-011.

7. Laat h_n gelijk zijn aan het product van de positive cijfers van n in het g-tallig stelsel tot de macht k. Dan is het getal $\sum_{n=1}^{\infty} \frac{h_n}{n!}$ irrationaal.

8. Zij $f = f_c$, $c \in Z$, een multiplicatieve functie met $f_c(p) = p + c$ for elk priemgetal p. Een f-bevriend paar is een tweetal positieve gehele getallen (a_1, a_2), $a_1 \neq a_2$, dat voldoet aan $f(a_1) = f(a_2) = (a_1 + a_2)/k$ voor een $k \in \mathbb{N}$. Stel dat (au, ap) een f-bevriend paar is, met p een priemgetal dat a niet deelt. Dan is ook (auq, ars) een f-bevriend paar, wanneer r, s en q onderling verschillende priemgetallen zijn met ggd$(au, rsq) = 1$ zodanig dat $(r - p)(s - p) = f(p)(p + u)$ en $q = r + s + u$.

   Dit is een generalisatie van de regels voor $f = \sigma$ (som van de delers-functie) en $f = \phi$ (Euler's ϕ-functie), zoals gegeven, respectievelijk, in:

   Herman J.J. te Riele. New very large amicable pairs. In H. Jager, uitgever, *Number Theory Noordwijkerhout 1983*, pages 210–215. Springer, 1984.

   Graeme L. Cohen and Herman J.J. te Riele. On ϕ-amicable pairs. *Math. Comp.*, 67(221):399–411, 1998.

9. Ook sommige niet-elektronische stemsystemen waarborgen het stemgeheim voor de 'kenner' niet, zoals het voorbeeld van ballotage, beschreven door Tolstoi in 'Anna Karenina', vertaald uit het Russisch door Huisman in de volgende passages:

   Uit deel 6, hoofdstuk 28

   > ... **Lewin kleurde, stak haastig zijn hand onder het laken en legde het balletje rechts neer, daar hij het in zijn rechterhand had. Terwijl hij het deed, bedacht hij, dat hij er ook zijn linkerhand in moest steken en hij deed het, maar het was al te laat en hij werd nog verlegener en trok zich zo snel mogelijk terug in de achterste rijen.**
   >
   > ...

   Uit deel 6, hoofdstuk 30

   > ... Toen hij bij de stembus kwam, had hij zijn balletje in de rechterhand, maar in de overtuiging, dat het verkeerd was, nam hij het juist voor de bus in zijn linker en legde het dus links neer. **Een kenner op dit gebied, die naast de kist stond en alleen al aan de beweging van de elleboog zag hoe ieder stemde, fronste onwillekeurig zijn voorhoofd. Hij had zijn scherpzinnigheid hierbij niet nodig gehad.**
   >
   > ...

   L.N. Tolstoj. *Anna Karenina.* Rainbow Pocketboeken. Maarten Muntinga bv, Amsterdam, 1995. Vertaling: Wils Huisman.

10. Een andere vertaler van 'Anna Karenina', Wasiltsjikow, geeft een positiever beeld van het stemsysteem door het eerste vetgedrukte stuk in de vorige stelling te vertalen met

    > Blozend en verlegen deponeerde Lewin op goed geluk zijn balletje onder het laken.

    en het andere vetgedrukte stuk weg te laten.

    Leo Tolstoi. *Anna Karenina.* Bigot & Van Rossum b.v., Blaricum, 14de druk. Vertaling: A. M. Wasiltsjikow.

11. De gebruikelijke treinstellen van de Nederlandse Spoorwegen zijn niet voorzien van gordijnen of jalouzieën. Dit duidt erop dat in Nederland de zon minder schijnt dan in landen waar de meeste treinen wel over zonweringen beschikken. Het feit dat sommige heel oude treinstellen van de Nederlandse Spoorwegen nog wel zonwering bieden suggereert dat de zon over de laatste decennia minder is gaan schijnen.

12. Terwijl in Nederland mensen 'achter de computer' zitten, zitten mensen in Duitsland 'vor dem Computer'. Niettemin blijken de mensen in beide landen wel degelijk aan dezelfde kant van de computer te zitten.