KUNGL TEKNISKA HÖGSKOLAN
Royal Institute of Technology
Dept. of Numerical Analysis and Computer Science

The number field sieve

An integer factorization algorithm

by
Johnny Bigert

TRITA-NA-E0053

NADA

Nada (Numerisk analys och datalogi)
KTH
100 44 Stockholm

Deptartment of Numerical Analysis
and Computer Science
Royal Institute of Technology
SE-100 44 Stockholm, SWEDEN

The number field sieve

An integer factorization algorithm

by
Johnny Bigert

TRITA-NA-E0053

Master's Thesis in Computer Science (20 credits)
at the School of Computer Science and Engineering,
Royal Institute of Technology year 2000
Supervisor at Nada was Johan Håstad
Examiner was Johan Håstad

Abstract

The topic of this Master's thesis is factorization of large integers using the *number field sieve*, the best factorization algorithm known today. We explain the theory behind the algorithm giving examples of the algebraic structures involved. We give a brief survey of an implementation of the algorithm, which is later used in the experiments.

We try to improve the speed of the implementation by experimenting with the *sieving area*, where we search for pairs (a, b) from which we form *relations*. The faster we find the relations, the faster the speed of the algorithm. We compare a rectangular sieving area to one following a plot of the "height curves" of the norm, i.e. curves where the norm is constant. We also use the definition of the norm to adapt the length of the a-interval, since different values of b have different probabilities to form relations. Furthermore, we investigate *smoothness* by approximating the smoothness of the norm as compared to the random numbers with a normal distribution.

Talkroppssållet

En algoritm för faktorisering av stora heltal

Sammanfattning

Denna avhandling behandlar faktorisering av stora heltal med *talkroppsållet*, den bästa kända faktoriseringsalgoritmen för närvarande. Vi förklarar teorin bakom algoritmen och ger exempel på de algebraiska strukturer som används. Vi ger en kort genomgång av en implementation av algoritmen som sedan används i experiment.

Vi försöker förbättra hastigheten på implementationen genom experiment med *sållningsytan*, där vi söker efter par (a, b) från vilka vi bildar *relationer*. Ju snabbare vi kan finna relationerna, desto snabbare blir algoritmen. Vi jämför en rektangulär sållningsyta med en som följer de "höjdkurvor" som bildas då man plottar en konstant storlek på normen. Vi använder även definitionen av normen för att anpassa längden av a-intervallen eftersom olika värden på b har olika sannolikhet att bilda relationer. Slutligen undersöker vi *glatthet* genom att approximera glattheten hos normen i jämförelse med slumptal med en normalfördelning.

Acknowledgments

First of all, I would like to thank my supervisor, professor *Johan Håstad* for his always clear-sighted considerations and suggestions for improvement. Without them, I would not have learnt half of what I did.

I am very grateful to my roommates, *Olof Åsbrink* who helped me with various TEX questions and *Joel Brynielsson*, who expressly wanted to get into the acknowledgments for his TEX knowledge. I got my practical knowledge of the number field sieve programs with the help of *Gunnar Andersson, Staffan Ulfberg, Lars Ivansson* and Johan Håstad. They were always there when I had topics I wanted to discuss. Also thanks to *Torbjörn Granlund* who made me state my knowledge explicitly on the whiteboard. I had much use of his math package `GMP`. Thanks to TEX master *Lars Engebretsen* for explaining to me why my report looks the way it does. Thanks also to the other PhD students and the staff at Nada who are always nice and helpful.

Thanks to my friends *Jakob Scott, Rickard Arnerup* and *Niclas Odljung* for reminding me that I was writing on my thesis every day, *Niclas Holm* for his support in every way and my father, *Sten Bigert*, for proofreading an endless amount of reports. To all the people that I have forgotten here: I haven't forgotten you!

Contents

Chapter 1

Introduction

This Master's thesis is about the factorization of large integers and specifically, the number field sieve. In this chapter, we will say something about integer factorization and give a small example.

1.1 Prime and composite numbers

We define an integer to be *prime* if it is divisible only by itself and one. All numbers that are not prime are called *composite*. For example, 5 is a prime while $6 = 2 \cdot 3$ is composite. We call each prime divisor of a number a *factor*. The factors of a composite number are unique if we ignore the ordering of the factors.

The *factorization* of a number is the procedure of determining its factors. For a long time, the factorization of numbers was of mere theoretical interest. It became of practical interest when Rivest, Shamir and Adleman introduced the RSA public-key cryptosystem [1] in 1978, which is based on the belief that factorization is a hard computational problem. In other words, the factorization of a number takes a huge amount of computer resources if the number is large.

1.2 Factorization methods

Even though the factors of a composite number exist and are uniquely determined, we have no way of determining them by looking at the number. We have to use some *algorithm*, or step-by-step computing procedure.

If we have a range of numbers which we want to factor, we can use a procedure called a *sieve*, originating from the *sieve of Eratosthenes* (see for example [6]). Taking one prime p at the time, we note that it is a factor of every pth number. When this has been done for all primes we have found the factorization of each and every number.

To determine the factors of an arbitrary number N, there are two classes of algorithms. There are those which have a running time that depends on the size of the factors and those which have a running time depending only on the size of N.

In the first class we find the most naive factoring algorithm, namely the *trial division* algorithm. It simply tries to divide N with one prime at a time. Since the second largest prime divisor of N can be of size $\sqrt{N}$, the running time is $O(\sqrt{N})$. Furthermore, in the first class, we find the *Pollard-ρ* [11] and the $p-1$ [11] factoring algorithms which both have better characteristics than trial division. For example, Pollard-ρ runs in $O(\sqrt{p})$ where p is the second largest prime divisor. The algorithms of this class are used to find small factors of N since the running times depend on the size of the factors found.

In the second class we have, for example, the *multiple polynomial quadratic sieve* [12] and the *number field sieve* [8]. They are both based on the idea of finding two integers x and y such that $x^2 \equiv y^2 \bmod N$ which gives half a chance that $\gcd(x - y,\ N)$ is a factor of N. These two algorithms have better characteristics than those of the first class and are used when no more small factors exist.

1.3 An example of integer factorization

To clarify the ideas of the previous section, we will factor an example number using $x^2 \equiv y^2 \bmod N$. We choose $N = 143$ and want to determine its factors.

A first approach could be to simply write squares modulo 143, as follows:

$$\begin{aligned}
12^2 &\equiv 1 = 1^2 \\
13^2 &\equiv 26 = 2 \cdot 13 \\
14^2 &\equiv 53 \\
15^2 &\equiv 82 = 2 \cdot 41 \\
16^2 &\equiv 113 \\
17^2 &\equiv 3 \\
18^2 &\equiv 38 = 2 \cdot 19 \\
19^2 &\equiv 75 = 3 \cdot 5^2 \\
20^2 &\equiv 114 = 2 \cdot 3 \cdot 19.
\end{aligned}$$

In this case, we found squares on both sides in the first step. We get that $\gcd(12 - 1) = 11$, which is a factor of $143 = 11 \cdot 13$. Normally, we would not be so lucky and we might have to search for a long time. Instead, if we observe the prime factors of the right hand side, we see that some of the factors are found in more than one row. For example, if we multiply 17^2 with 19^2 we get that $17^2 \cdot 19^2 \equiv 3 \cdot 3 \cdot 5^2$, which is square on both sides! We

s	s mod 143	2	3	5	7	-1	2	3	5	7
$128 = 2^7$	$-15 = -3 \cdot 5$	7				1		1	1	
$135 = 3^3 \cdot 5$	$-8 = -2^3$		3	1		1	3			
$150 = 2 \cdot 3 \cdot 5^2$	7	1	1	2						1
$168 = 2^3 \cdot 3 \cdot 7$	$25 = 5^2$	3	1		1				2	
$175 = 5^2 \cdot 7$	$32 = 2^5$			2	1		5			
$288 = 2^5 \cdot 3^2$	2	5	2				1			
$300 = 2^2 \cdot 3 \cdot 5^2$	$14 = 2 \cdot 7$	2	1	2			1			1
$336 = 2^4 \cdot 3 \cdot 7$	$50 = 2 \cdot 5^2$	4	1		1		1		2	

Table 1.1. *Factorizations of some numbers that have prime factors exclusively below 7 both before and after reduction modulo 143.*

get $\gcd(17 \cdot 19 - 3 \cdot 5, 143) = 11$. Another example would be to multiply $17^2, 18^2$ and 20^2.

To generalize the idea above, we use a technique involving matrices to determine which equations should be multiplied together. Given an integer s, we determine the factors of s and the factors of s mod N. For example: $336 = 2^4 \cdot 3 \cdot 7 \equiv 50 = 2 \cdot 5^2$. If we find even powers of the factors on both sides of the congruence, we have two squares with the desired property. This task becomes much easier if we choose to use only numbers having divisors below 7, for example, and we call these numbers 7-smooth.

One way to find 7-smooth numbers would simply be to factor every integer from one and up. We chose eight of these numbers: 128, 135, 150, 168, 175, 288, 300 and 336. The factorization of each of the numbers is shown in Table 1.1. Observe that we include -1 among the factors for the right-hand side.

To find a square on both sides we need even powers of the divisors. Therefore, we can add powers modulo 2. The matrix from Table 1.1 is given modulo 2 in Table 1.2. We see that 150, 288 and 300 yield the desired squares if multiplied together: $150 \cdot 288 \cdot 300 = (2^5 \cdot 3^2)(2 \cdot 3 \cdot 5^2)(2^2 \cdot 3 \cdot 5^2) = 2^8 \cdot 3^4 \cdot 5^4 = (2^4 \cdot 3^2 \cdot 5^2)^2 = 3600^2 \equiv 2 \cdot 7 \cdot 14 = (2 \cdot 7)^2 = 14^2 \bmod 143$. This gives us $\gcd(3600 - 14, 143)$=11.

Normally, one uses Gaussian elimination modulo 2 on the matrix to find the rows that are to be multiplied.

s	s mod 143	2	3	5	7	-1	2	3	5	7	
$128 = 2^7$	$-15 = -3 \cdot 5$	1				1		1	1		
$135 = 3^3 \cdot 5$	$-8 = -2^3$		1	1		1	1				
$150 = 2 \cdot 3 \cdot 5^2$	7	1	1							1	$\leftarrow$
$168 = 2^3 \cdot 3 \cdot 7$	$25 = 5^2$	1	1		1						
$175 = 5^2 \cdot 7$	$32 = 2^5$				1		1				
$288 = 2^5 \cdot 3^2$	2	1					1				$\leftarrow$
$300 = 2^2 \cdot 3 \cdot 5^2$	$14 = 2 \cdot 7$		1				1			1	$\leftarrow$
$336 = 2^4 \cdot 3 \cdot 7$	$50 = 2 \cdot 5^2$		1		1		1				

Table 1.2. *The divisors from table 1.1 given modulo 2. The rows with the arrows give a square in both s and s mod 143 if multiplied together.*

Chapter 2

Definition of problem

2.1 Background

Over the years, several algorithms for factorization have been proposed. The latest theoretical proposition was issued by Pollard [8] in 1993, when the number field sieve (NFS) was presented. The time complexity showed it to be the asymptotically fastest algorithm ever for factoring. Implementations of another algorithm, the multiple polynomial quadratic sieve (MPQS) [12], had been successful with integers up to around 128 decimal digits. The NFS was assumed to do much better than MPQS on larger numbers.

2.2 Purpose

As predicted, implementations of NFS did outperform all other algorithms when numbers around 130 decimal digits were factored. In 1999, an implementation of the NFS from CWI/Oregon [7] factored a 155 digit decimal number originating from RSA512. In December 1999, Nada received a copy of the CWI/Oregon implementation on which this Master's thesis is based.

The question posed here is: Can the number field sieve implementation be improved for speed? With greater speed, larger numbers can be factored. From a theoretical point of view the purpose of this Master's thesis has been to understand the NFS algorithm.

2.3 Outline of the Master's thesis

The report has been divided into two parts, theory and practice. The theoretical part contains a description of the algorithm. Because of the mathematical nature of the thesis, a great deal of time has been spent in understanding the aspects of the algorithm. The theory part is therefore work done by others, interpreted and understood by us.

The practical part makes use of the CWI/Oregon software to answer a number of questions concerning the theory and practice of the algorithm. To some extent, the practical part of the Master's thesis has been to understand the vast number of parameters of the software. To reduce the amount of time spent on the experiments, we implemented a sampler that performed only part of a full factorization. With the sampler we could estimate a few of the algorithm characteristics, such as total running time, for several settings of the algorithm parameters.

Chapter 3

Theoretical aspects

3.1 Description of the NFS

In this chapter, we describe how the number field sieve algorithm is used to factor large integers. The algorithm makes use of concepts from algebraic number theory to find the factors. In the NFS algorithm, the difficulty of the problem does not depend of the size of the factors, but only on the number to be factored.

In the following, some algebraic terminology is used and it is assumed that the reader is familiar with some basic algebra. All concepts given in *italics* when introduced are explained in the dictionary in Appendix A. For a thorough explanation of the terminology, we refer to [5] or [10].

3.1.1 Outline of the NFS algorithm

Here, a short description is given of all steps of the NFS algorithm. The individual steps, given in **bold**, will then be thoroughly explained in sections of their own, so that all concepts can be understood.

We are given a large integer N and want to determine its factors. To accomplish this, we want to find two integers x and y so that $x^2 = y^2 \bmod N$. This gives us $\gcd(x - y, N) = s$ and with probability one half, $s \neq 1$ and $s \neq N$ implying that s is a factor of N.

First, we choose two **polynomials** and $m \in \mathbb{Z}$ such that

$$f_1(m) = f_2(m) = 0 \bmod N. \tag{3.1}$$

If we take a complex root α of a polynomial, we can construct an **algebraic number field** $\mathbb{Q}(\alpha)$ and from that, an algebraic structure $\mathbb{Z}[\alpha]$. In $\mathbb{Z}[\alpha]$ we have elements on the form $\sum a_i \alpha^i$, $a_i \in \mathbb{Z}$. We also define a **homomorphism** $\phi : \mathbb{Z}[\alpha] \to \mathbb{Z}_N$ taking α to m, that is, $\phi : \sum a_i \alpha^i \to \sum a_i m^i$.

With these preliminaries we can define the objective of the algorithm,

namely to find γ_1 and γ_2 such that

$$(\phi_1(\gamma_1))^2 = \phi_1\Big(\prod_{(a,b)\in S}(a - b\alpha_1)\Big) = \phi_2\Big(\prod_{(a,b)\in S}(a - b\alpha_2)\Big) = (\phi_2(\gamma_2))^2 \tag{3.2}$$

where α_1 and α_2 are the roots of polynomials 1 and 2, ϕ_1 and ϕ_2 are the homomorphisms originating from polynomials 1 and 2, respectively, and all factors are **ideals**. The speed of the algorithm is determined by how fast we can find the set S of pairs (a, b), $a, b \in \mathbb{Z}$.

To simplify operations on the elements of $\mathbb{Z}[\alpha]$ when finding S, we will work with the **norm** of the number, which is in $\mathbb{Q}$, instead of the number itself. If an element of norm r is divisible by an element of norm s then s divides r. To simplify further, we choose a large integer as a bound of the norm of the divisors of γ. We call this large integer the **factor base** bound, to be explained later.

In the algorithm, we actually find two **squares** γ_1^2 and γ_2^2. From these, we extract the **square roots** to get γ_1 and γ_2. If we define $x = \phi_1(\gamma_1)$ and $y = \phi_2(\gamma_2)$, we get $x^2 = y^2 \bmod N$ and we have a factorization of N.

3.1.2 Polynomials

The first step of the NFS algorithm is to choose two polynomials with certain properties. From these, we create the algebraic structures used by the NFS.

We choose two polynomials f_1 and f_2, each having the greatest common divisor of the coefficients equal to one. We want the polynomials to have a common root modulo N, such that $f_1(m) = f_2(m) = 0 \bmod N$. They can be assumed to be irreducible in $\mathbb{Z}$ since if f is reducible it can be written $f = gh$, where either g is irreducible and can be chosen instead of f or $g(m)$ is a factor of N. The same applies to h. A polynomial is irreducible in $\mathbb{Z}$ if and only if it is irreducible in $\mathbb{Q}$ [5], so we assume that both f_1 and f_2 are irreducible in $\mathbb{Q}$.

Note here that a polynomial of degree d has at most d roots of which some are real and some are complex. The complex roots always appear in conjugate pairs. Note also that an equation $f(s) = 0 \bmod p$ has at most d integer roots between 0 and $p - 1$ if p is prime.

3.1.3 Algebraic number fields

From the polynomials of the previous section, we construct two *algebraic number fields*. The elements of these fields are used to form the algebraic structures on which the NFS algorithm is based.

Given $f(x)$, a polynomial without solutions in $\mathbb{Q}$, let α be a solution of $f(x)$ in an extension of $\mathbb{Q}$. We define an algebraic number field $\mathbb{Q}(\alpha)$ as the smallest ring containing both $\mathbb{Q}$ and α. Since α is a root of $f(x)$ in $\mathbb{Q}(\alpha)$ we have $f(\alpha) = 0$ and all operations can be done mod $f(x)$.

To simplify the exposition in this thesis, we will assume that the polynomials are monic, so that $\mathbb{Z}[\alpha]$ introduced below is actually a *number ring*. If the polynomial is not monic and $\mathbb{Z}[\alpha]$ is not a ring, we refer to [8] for a solution to this problem.

From the algebraic number field, we construct the number ring $\mathbb{Z}[\alpha]$ mentioned above, which is all polynomials in α with coefficients in $\mathbb{Z}$. As in $\mathbb{Q}(\alpha)$, all higher powers of α are reduced using $f(\alpha) = 0$. The number ring $\mathbb{Z}[\alpha]$ does not necessarily have unique factorization since there might exist elements that factor into *irreducibles* in more than one way. As an example, if we take $\mathbb{Q}(\sqrt{10})$ and the ring $\mathbb{Z}[\sqrt{10}]$, we have that $6 = 2 \cdot 3 = (4+\sqrt{10})(4-\sqrt{10})$, where all factors are irreducible (not shown here). If the number ring $\mathbb{Z}[\alpha]$ does not have unique factorization, this can be resolved as explained in the next section.

For the NFS algorithm, we use the polynomial f and choose one arbitrary root $\alpha \in \mathbb{C}$. Then, α generates an algebraic number field $\mathbb{Q}(\alpha)$ and we get our ring $\mathbb{Z}[\alpha]$. This is done for both polynomials, giving us two number rings.

3.1.4 Ideals

From the previous section we saw that unique factorization could not always be obtained with the algebraic numbers in a ring $R = \mathbb{Z}[\alpha]$. To resolve this, we introduce *ideals* that are subsets generated by one or more algebraic numbers. The ideals are closed under multiplication and addition with elements of the ring. We first explain how to calculate with ideals before explaining how unique factorization is obtained.

The ideal $\mathcal{I}$ generated by the algebraic numbers $\beta_1, \beta_2, \ldots, \beta_n \in \mathbb{Z}[\alpha]$ contains $\beta_1 \cdot \mathbb{Z}[\alpha] + \beta_2 \cdot \mathbb{Z}[\alpha] + \ldots + \beta_n \cdot \mathbb{Z}[\alpha]$ and is denoted by $[\beta_1, \beta_2, \ldots, \beta_n]$. An ideal generated by one algebraic number β is called a principal ideal and is denoted by (β).

Multiplication of two ideals $\mathcal{I}$ and $\mathcal{J}$ gives a new ideal that contains all finite sums of products of elements of $\mathcal{I}$ and $\mathcal{J}$. Division is defined as follows: if $\mathcal{I}$ divides $\mathcal{K}$ then there exists an ideal $\mathcal{J}$ such that $\mathcal{K} = \mathcal{I}\mathcal{J}$. From the definition of multiplication we can deduce that division is equivalent to containment.

Norm, prime ideals and factorization

We define the norm of an ideal $\mathcal{Q}$ as $|\mathbb{Z}[\alpha]/\mathcal{Q}|$. For example, if we take $\mathcal{Q} = [3, 1+\sqrt{10}]$ in $\mathbb{Z}[\sqrt{10}]$ all elements are on the form $3(c_1 + c_2\sqrt{20}) + (1 + \sqrt{10})(d_1 + d_2\sqrt{10}) = a + b\sqrt{10}$. To determine which numbers are in $\mathcal{Q}$ we solve the linear equation system:

$$\begin{array}{rcrcrcl} 3c_1 & & + & d_1 & + & 10d_2 & = & a \\ & 3c_2 & + & d_1 & + & d_2 & = & b \end{array}$$

By subtracting the first equation from the second, we get $3(-c_1+c_2-3d_2) = b-a$, which is solvable if and only if $b-a = 0 \bmod 3$. It can be seen that every number in $\mathbb{Z}[\sqrt{10}]$ can be written on the form $3(c_1 + c_2\sqrt{20}) + (1 + \sqrt{10})(d_1 + d_2\sqrt{10}) - k$, which gives us $3(-c_1 + c_2 - 3d_2) - k = b - a$ where $k = 0, 1, 2$. Thus, we have three cosets in $\mathbb{Z}[\alpha]/\mathcal{Q}$ and the norm is $N(\mathcal{Q}) = 3$.

A prime ideals is an ideal divisible only by itself and the entire ring. Also, an ideal is prime if and only if it has a prime norm. For example, $\mathcal{Q}$ above is a prime ideal since 3 is a rational prime. The ideals have unique factorization into prime ideals [10].

Example of factorization into prime ideals

As an example of ideal factorization, we use the counterexample of unique factorization in $R = \mathbb{Z}[\sqrt{10}]$ from Section 3.1.3. We have that $6 = 2 \cdot 3 = (4+\sqrt{10})(4-\sqrt{10})$, where all factors are irreducible in R. From the definition, $(2) = 2\mathbb{Z} + 2\mathbb{Z} \cdot \sqrt{10}$ and $(3) = 3\mathbb{Z} + 3\mathbb{Z} \cdot \sqrt{10}$. If we define $\mathcal{P} = [2, \sqrt{10}]$, $\mathcal{Q} = [3, 1 + \sqrt{10}]$ and $\mathcal{Q}' = [3, -1 + \sqrt{10}]$, which are all prime ideals, we get a prime ideal factorization $(2) = \mathcal{P}^2$ and $(3) = \mathcal{Q}\mathcal{Q}'$. Thus, $(6) = \mathcal{P}^2\mathcal{Q}\mathcal{Q}'$ and this factorization is unique up to the ordering of the factors. We see that $4 + \sqrt{10}$ is in $\mathcal{Q}$ since $4 + \sqrt{10} = 3 \cdot 1 + (1 + \sqrt{10}) \cdot 1$ and similarly, $4 - \sqrt{10}$ is in $\mathcal{Q}'$. Since $\mathcal{Q}$ contains the generator $4 + \sqrt{10}$ of $(4 + \sqrt{10})$, we see that $\mathcal{Q}$ contains $(4 + \sqrt{10})$ and therefore, $\mathcal{Q}|(4 + \sqrt{10})$. By calculating the norms (not shown here) $N(\mathcal{P}) = 2$, $N(\mathcal{Q}) = 3$ and $N((4 + \sqrt{10})) = 6$ and using the fact that the norm is multiplicative, we deduce that $(4 + \sqrt{10}) = \mathcal{P}\mathcal{Q}$ and similarly, $(4 - \sqrt{10}) = \mathcal{P}\mathcal{Q}'$.

3.1.5 Norm of an algebraic number

The algebraic numbers are rather complex to work with, and instead we use the *norm* of the number which have many good properties. This allows us to work with rational numbers instead of algebraic numbers.

For the sake of completeness, we give the definition of the norm. The norm of an algebraic number $\sum_i a_i\alpha^i$ is defined as the product of the *embeddings*, that is, $\prod_k (\sum_i a_i\theta_k(\alpha)^i)$, where α is a root of the polynomial.

In the NFS, we are interested only in the numbers on the form $a - b\alpha$. They have a simple structure of their norms and a small absolute norm value. Let d be the degree of f. We define

$$F(a, b) = b^d f(a/b), \tag{3.3}$$

a polynomial in $a \in \mathbb{Z}$ and $b \in \mathbb{Z}$ which is integer valued. The norm turns out to be $N(a - b\alpha) = F(a, b)/c$, where c is the leading coefficient of f. Since our polynomials need not be monic, the norm is a rational number. Furthermore, the norm is multiplicative, that is, $N(\alpha\beta) = N(\alpha)N(\beta)$.

The norm of an ideal generated by an algebraic number γ satisfies

$$N((\gamma)) = |N(\gamma)|,$$

where the norm on the left is the norm of an ideal and the norm on the right is the norm of an algebraic number. Thus, when calculating with numbers on the form $a - b\alpha$ we use the norm of the algebraic number instead of the ideal to simplify the calculations.

3.1.6 Square algebraic numbers

The objective of the NFS algorithm is to find two squares γ_1^2 and γ_2^2 of algebraic numbers, one from each of the number rings from Section 3.1.3. We want to find a set S of relations such that

$$(\gamma_j^2) = \prod_{(a,b)\in S} (a - b\alpha_j), \tag{3.4}$$

where $j = 1, 2$ for the two number rings, respectively, and all factors involved are ideals.

To find a square, all powers of the divisors must be even. The divisors are prime ideals, which are chosen from a limited set described in the next section. With a limited set of prime ideals, a method of finding the members of S can be constructed. This method is explained in Section 3.1.8.

3.1.7 The factor base

We limit the ideals we work with in the algorithm to a set of prime ideals called the factor base, by determining an upper bound on the norms of the ideals.

We choose a large integer value B_j, called the lower factor base bound, for each of the two polynomials. We also choose an integer $L_j \geq B_j$ called the upper factor base bound. We want to find a set S of (a, b)-pairs according to equation 3.4, where all factors in γ_j have norm below B_j, except perhaps for a few large primes between B_j and L_j. Allowing a few large primes to be factors of the norm greatly increases the probability that a pair (a, b) is included in S.

To simplify the exposure in this section, we will assume that $B_j = L_j$ for both polynomials. In the next section, we explain how these values are used in the algorithm.

To see how the choice of B_j affects the factorization of γ_j we look at how the roots of the polynomials are used to identify prime ideal divisors of $a - b\alpha$.

For every prime $p \leq B_j$ and r between 0 and $p - 1$, we identify each of the solutions of $f_j(r) = 0 \bmod p$ with a pair (p, r), and we can have at most degree(f_j) pairs for each p. The prime ideals of $\mathbb{Z}[\alpha]$ of norm p are in a

one-to-one correspondence with the pairs (p, r). That is, given a prime ideal $\mathcal{P}$, the map

$$\mathbb{Z}[\alpha] \to \mathbb{Z}[\alpha]/\mathcal{P} \cong \mathbb{Z}/p\mathbb{Z} \tag{3.5}$$

takes α to r mod p and we have that p and $r - \alpha$ generate the ideal $\mathcal{P}$. Furthermore, the element $\sum s_i \alpha^i$ is in $\mathcal{P}$ if and only if $\sum s_i r^i = 0 \mod p$, and from this we conclude that if $a - b\alpha$ is in $\mathcal{P}$ then

$$a - br = 0 \mod p. \tag{3.6}$$

This only happens for one pair (p, r), giving us a method of determining to which prime ideal the full power of p belongs.

Now, if we have $\gamma \in \mathbb{Z}[\alpha]$ of norm M and p such that $p|M$, we know that one of the prime ideals corresponding to a pair (p, r) divides γ. Our factor base is made out of these prime ideals. In numbers, there are approximately as many pairs as there are prime numbers below B_j, since each of the $f(r) = 0 \mod p$ has one solution per p on average.

3.1.8 Finding a square

As stated in section 3.1.6 we want to find a square algebraic number. This is done with the factor base and some linear algebra.

In short, we want to construct two squares by assuring that every factor appears an even number of times. This can be done by introducing a matrix into which we insert the relations, and from the matrix we find a linear dependency modulo 2. These steps are explained below.

The matrix

We define a relation as a pair (a, b) for which $F(a, b)$ is B_j-*smooth* except perhaps for a few large primes between L_j and B_j.

Let each relation found constitute a row in a matrix, and let the columns be the factor base elements. We have that the norm of each relation is a product of primes $\leq B_j$ and perhaps some large primes that we will deal with later. Specifically, each prime ideal identified by equation (3.6) corresponds to a column in the matrix, see Figure 3.1. If a, b and p satisfies (3.6), that is, $a = br \mod p$, we set the corresponding matrix entry to the largest k such that $p^k | F(a, b)$. For each relation, this will fill the matrix with the powers of the divisors of the norm.

Quadratic characters

One problem with the norm is that a square norm does not imply that the algebraic number is square. This is because the units have norm ± 1, which is square even though the unit itself need not be a square. As an example, take $\mathbb{Z}(\sqrt{3})$ generated by the polynomial $f(x) = x^2 - 3$, and consider the element

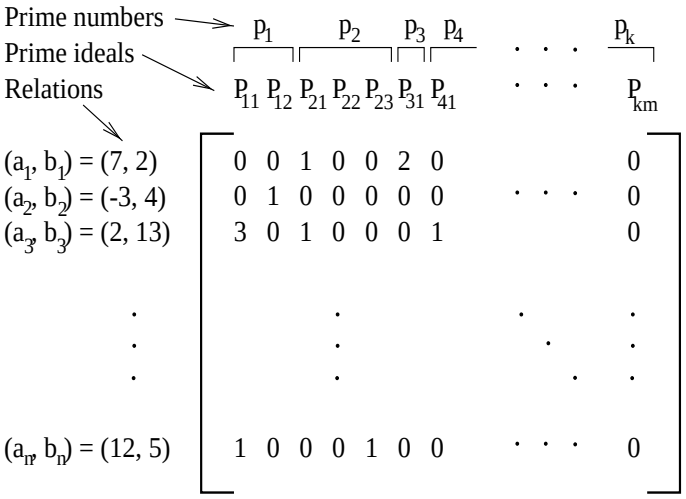

Figure 3.1. *An example of the matrix built from the relations found. The prime numbers are two sets bounded by B_1 and B_2, respectively. The prime ideals are identified with the solutions of $f(r) = 0$ mod p, so that several prime ideals can belong to a prime number. For each relation, at most one prime satisfies $a = br$ mod p, and therefore identifies precisely one prime ideal. For this r, the matrix entry is the largest k such that p^k divides the norm of the relation.*

$2 + \sqrt{3}$. We have that $N(2 + \sqrt{3}) = F(2, -1)/c_2 = (-1)^2((\frac{2}{-1})^2 - 3)/1 = 4 - 3 = 1$, which might lead us to believe that $2 + \sqrt{3}$ is a square, but if we solve $x^2 = 2 + \sqrt{3}$ we get that $x = \pm(\sqrt{6} + \sqrt{2})/2$ which is not in $\mathbb{Z}[\sqrt{3}]$! We deal with the unit problem by introducing so called quadratic characters.

We first observe that an integer cannot be a square if it is not square modulo some given prime. The quadratic characters use this fact to ensure that an algebraic number is square.

We make use of the Legendre symbol, defined as $(\frac{c}{p})$ equals 1 if $x^2 = c$ mod p is solvable and -1 otherwise. We want to determine whether $a - b\alpha$ is square or not. We take a large prime q and find a number $r < q$ such that $f(r) = 0$ mod q. From this, we define a homomorphism $\phi : \mathbb{Z}[\alpha] \to \mathbb{Z}_q$ taking α to r. If $a - b\alpha$ is a square, i.e. $a - b\alpha = \gamma^2$, we have that

$$\phi(a - b\alpha) = \phi(\gamma^2) = \phi(\gamma)^2 = x^2 = a - br, \tag{3.7}$$

where the last equality comes from $\phi(a - b\alpha) = a - br$. If $(\frac{a-br}{q}) = -1$, we know that $a - br$ is not square modulo q and no x satisfies (3.7). Therefore, $a - b\alpha$ can not be square. If $(\frac{a-br}{q}) = 1$, we get no information. If we repeat the calculations for many q and all results are 1, there is a good probability that we have a square.

The results from the different values of q are augmented to the matrix

rows as a 0 if $(\frac{a-br}{q}) = 1$ and a 1 otherwise. With this additional information the relation (a, b) is most probably square if all quadratic characters are 0.

Linear dependencies

Our only interest is if the product of our relations is a square, hence we want to determine a set of relations such that the product gives us a row with no odd element. Therefore, all calculations can be done modulo 2. We aim at finding at least as many relations as there are members in the factor bases so that we are assured that a linear dependency can be found. We then have a square of algebraic numbers in both factor bases and we call these γ_1^2 and γ_2^2.

Primes above the factor base bound

If we observe the integers between B_j and L_j we see that some will occur in more than one relation. These relations can form new relations, if we seek to match the large primes in pairs.

To convince ourselves that there exist prime ideals $\mathcal{P}$ that correspond to each of the occurring large primes, we look at p and a/b. We see that r is uniquely determined from (3.6), so that $\frac{a}{b} = r \bmod p$. Then the pair (p, r) uniquely identifies a prime ideal as explained in Section 3.1.7.

Free relations

If the polynomial $f(x) = 0 \bmod p$ factors completely over $\mathbb{Z}$, we have precisely degree(f) roots, counted with multiplicity. We then have degree(f) prime ideals, that correspond to the roots. From (3.6) we see that since $a - br_k = 0 \bmod p$ is fulfilled for each of the roots, we have that each of the prime ideals $\mathcal{P}_k$ corresponding to (p, r_k) is a divisor of the principal ideal $\mathcal{P}$ generated by p. We also know that $F(p, 0) = p^d$, so that $\mathcal{P}$ can have no more than degree(f) divisors of norm p. Then $\mathcal{P} = \prod_{k=1}^{d} \mathcal{P}_k$, and we get a relation for free in the matrix, namely $(a, b) = (p, 0)$ with powers corresponding to the multiplicity of each root.

Then, how often does $f(x) = 0 \bmod p$ have degree(f) solutions? The Galois group of a polynomial f is the group of permutations of the zeros of f in $\mathbb{R}$. From Galois theory, it is known that the answer to the question is $1/g$ [13], where g is the order of the Galois group of f. With two polynomials we have that $\frac{1}{g_1 g_2}$ of the primes would give rise to a free relation, where g_1 and g_2 are the order of the Galois groups of each polynomial, respectively. Most often, the group of permutations of a polynomial with degree d is the symmetric group S_d, i.e. all permutations of length d. For example, with two such polynomials of degree d_1 and d_2 respectively, we get that approximately $\frac{1}{d_1! d_2!}$ of the primes should give a free relation.

3.1.9 Square roots

The next step of the NFS is to extract the square root of γ_j^2 to get γ_j. The square root algorithm is not covered here but a few remarks can be made. As the matrix is filled with the exponents of each prime ideal, it would seem as an easy task to divide all exponents by two to get the square root, but the occurrence of units makes this infeasible. Also, the fact that ideals are not principal poses a problem.

The algebraic number γ_j^2 is a product of a vast number of factors on the form $a - b\alpha_j$, and written as a polynomial in α_j, its coefficients are enormous. Therefore, it is not feasible to calculate the coefficients and from them compute the square root. Instead, Couveignes [4] proposed a method that calculates the root modulo a few primes and then applies the Chinese remainder theorem. Yet another method was given by Montgomery [7], which puts part of the square in the numerator and part in the denominator and iteratively extracts factors from the quotient. The algorithms aim at finding the prime ideal divisors of γ, the result is expressed as a set of generators of the prime ideals, also including those occurring from the primes between B_j and L_j.

3.1.10 The homomorphism

When the two square roots are found we need a way to map the algebraic numbers to the integers modulo N. We therefore use a natural *homomorphism* $\phi : \mathbb{Z}[\alpha] \to \mathbb{Z}/N\mathbb{Z}$ given by $\phi(\sum_i a_i\alpha^i) \to \sum_i a_i m^i$, where m is the common root of the polynomials from Section 3.1. Simplified, one can see this evaluation homomorphism as replacing α with m. For example, if we have $m = 7$ then $\phi(2 + 3\alpha) = 2 + 3 \cdot 7 = 23$.

We apply the homomorphisms ϕ_1 and ϕ_2 from polynomials 1 and 2, respectively, yielding $\phi_1(\gamma_1)^2 = \phi_1(\gamma_2)^2 \bmod N$. With a probability of one half, we get that s=gcd$(\phi(\gamma_1) - \phi(\gamma_2), N) \neq 1$ or N and we have that s is a non-trivial factor of N.

Chapter 4

Practical and implementational aspects

4.1 Introduction

In this chapter we describe the different parts of the NFS implementation from CWI/Oregon. To determine if the algorithm can be improved for speed, we do experiments with the norm and with the set from which we choose the relations.

4.2 The CWI/Oregon software

The implementation from CWI/Oregon is divided into several programs, each dedicated to a specific stage of the NFS algorithm. We list them here and give a short description. The programs are then described in sections of their own. The running times below are given for a 155 decimal digits number.

Polynomial finder The polynomial finder searches for two polynomials with good characteristics. It keeps improving the polynomials found and must be stopped manually when the characteristics are satisfactory. Typical running times are CPU weeks. The program can be run in parallel.

Root finder The root finder is run once to prepare for the sieve. Typical running time is CPU hours. The implementation can not be run in parallel.

Sieve The sieve finds the relations. Running time is CPU years and the program is highly parallel.

Relation post processing The relations are filtered to eliminate duplicate and useless relations from the sieve. The program is run in serial and takes about one CPU day. The memory demand is gigabytes.

Matrix and linear dependency The relation is inserted into a matrix from which a linear dependency is found. Running time to find a linear dependency can be CPU weeks and can not be run in parallel.

Square root From the linear dependency, two square algebraic numbers are found. The square roots are computed using a serial program. Running time is about one CPU day.

4.2.1 Polynomial finders

Two polynomial finders have been implemented. The first is based upon an idea by Montgomery [7] that finds two quadratic polynomials. The second is from 1996 and finds one polynomial of degree four or five and one linear polynomial. All tests in this report are performed with the second program.

When choosing the polynomials, the main objective is to keep the norm as small as possible. This is accomplished using the following ideas:

a) The coefficients ought to be chosen so that the norm is as small as possible over the sieving area (the set from which we choose the (a, b)-pairs).

b) The polynomials ought to have many solutions to $f(r) = 0 \bmod p$ for small primes. This reduces the size of the norm when dividing out the small primes as explained in Section 4.2.3.

c) The leading coefficient ought to have many small factors. This is an inherent property of the norm and is explained in the sieve section.

d) The gcd(*leading coefficient, second coefficient squared*) should be made as large as possible. This is closely related to the above and explained later.

e) The polynomials ought to have many small real roots. This is because the nearer the quotient a/b is to a real root, the smaller the value of the norm from equation (3.3).

4.2.2 Root finder

The root finder is not very interesting, it simply saves the solutions of $f_j(r) = 0 \bmod p$ for all primes $\leq B_j$ for both polynomials. This precalculation saves a lot of initialization time for the sieve.

4.2.3 The sieve

The sieve is used to find the required number of relations as quickly as possible.

We choose our relations from a set called the sieving area, originally chosen to be a rectangle with a-values on one side and b-values on the other. The sieving area has been subject to experiments, as explained in Section 4.3.

Description

We are interested in whether p divides $F(a, b)$. This depends only on (a, b) mod p, and we observe only coprime pairs of a and b. We get two cases, one where $b \neq 0 \bmod p$ and one where $b = 0 \bmod p$.

If $b \neq 0 \bmod p$, we know from (3.6) that the prime ideal corresponding to (p, r) divides $a - b\alpha$ if $a = br \bmod p$. This means that

$$p \text{ divides } F(a, b) \text{ if } a = br \bmod p. \tag{4.1}$$

On the other hand, if $b = 0 \bmod p$, then b^{-1} does not exist so $\frac{a}{b}$ has no meaning. We then turn to the definition of the norm. Let d be the degree of the polynomial f, yielding

$$F(a, b) = b^d f(a/b) = c_d a^d + c_{d-1} a^{d-1} b + c_{d-2} a^{d-2} b^2 + ... + c_0 b^d. \tag{4.2}$$

We see that, if $b = 0 \bmod p$, then $F(a, b) = c_d a^d \bmod p$. We then have that

$$\text{if } p|b \text{ and } p|c_d \text{ then } p \text{ divides } F(a, b) \text{ for all values of } a. \tag{4.3}$$

Observe that p can not divide a in this case since a and b are coprime.

Implementation

In the CWI/Oregon implementation the sieving is done by representing the norms by an approximation of its logarithm. When calculating with logarithms, division turns into subtraction which is favorable in a computational sense.

One b is chosen and a large array of a-values is initialized with the logarithm of each norm minus the logarithm of $\gcd(b, c_d)$ as implied by (4.3). Then, for each prime $p \leq B_j$ we find a pair (a, b) satisfying $a = br \bmod p$, and we step p, subtracting $\log(p)$ for each array element as shown by (4.1). In this manner, the logarithm of all factors below B_j is subtracted from each norm in the array.

In the CWI/Oregon software, this data type has eight bits so that a large number of norms fit into the CPU caches.

Coefficients of the polynomials

The norm can be made more likely to be smooth if the coefficients of the polynomials are chosen as explained in this subsection. If the leading coefficient c_d has square divisors p^2 and p divides the second coefficient c_{d-1}, then we conclude that if $p|b$ then p^2 divides $F(a,b)$ as seen from (4.2). Of course, this reasoning extends to higher powers of p, if the lower coefficient has the appropriate divisors.

Because of the symmetry between a and b, we see that the same applies to a and the lowest coefficient. If s=gcd(a, *constant coefficient of the polynomial*) is big, this leads to additional smoothness of the norm since s divides $F(a,b)$ for all values of b.

Finding the factors of the norm

When all factors are subtracted, the result of each array element is below a small threshold if the norm is B_j-smooth. If a few large primes between B_j and L_j are allowed, we choose a larger threshold. As an example, if we allow two large primes, a suitable choice is $\log(L_j^2)$. This is the case, since after dividing out all primes below B_j we will either have two prime factors or one large prime factor, and all factors found are between B_j and L_j as desired. If we allow three large primes and take the threshold $\log(L_j^3)$, the situation is more complicated since all factors do not necessarily fall between B_j and L_j.

When a relation has been marked as a candidate for being L_j-smooth, the norm will be factored. The factors below B_j can be found with the two criteria (4.1) and (4.3). If the remainder is not a prime, it is factored. A relation that has an L_1-smooth norm for the first polynomial f_1 and an L_2-smooth norm for the second polynomial f_2 is filed and later included in the matrix. We consider ourselves to be done when the number of relations is slightly more than the number of elements in the factor bases occurring in the relations.

4.2.4 Relation manipulation

After the sieve, we filter the relations to eliminate duplicate relations and relations which are too heavy, i.e. have many factors and will therefore give a less sparse matrix. The filter also finds singleton elements in the factor base, that is, prime ideals only occurring once among the relations. It also groups together relations containing factor base elements that can not be used otherwise, for example, elements only occurring twice in all relations found.

4.2.5 Finding a linear dependency

From the relations, a matrix is built using each of the remaining relations as rows and each of the occurring factor base elements as columns. The filtering stage has assured that the matrix is as sparse as possible. Since we are only interested in whether the result is a square or not, all manipulation of the matrix is done modulo 2.

As the matrix is built, we introduce the quadratic characters as explained in Section 3.1.8. We choose a set of q's and, for each relation, we append a 0 if the Legendre symbol is 1, and append a 1 otherwise. When the linear dependency is found, the quadratic characters are all zero, which implies that the algebraic number is likely to be square. The CWI/Oregon software uses 32 quadratic characters.

Normally, one would use Gaussian elimination to find a linear dependency, but with the size of the matrices it is not possible to store all the matrix elements in memory, even if each element is represented by one bit. Instead, the block Lanczos algorithm [3] is used, which is an iterative algorithm that updates one row at the time, not keeping the full matrix in memory.

When finding a linear dependency, we want to find coefficients $\{x_i\}$ such that $\sum_i x_i a_i = 0$, where the a_i's are column vectors from our matrix A. In matrix notation, we want to solve the equation $Ax = 0$. Unfortunately, given to the block Lanczos, this yields the trivial solution $x = 0$. Instead, we choose a random vector y and form $b = Ay$, then solve $Az = b$. Since $A(y + z) = 0$, we find that $x = y + z$, where all operations are modulo 2. Not all linear dependencies give rise to a factor of N, so we want to find at least a few solutions to the linear equations.

4.2.6 Extracting the square root

The square root algorithm implemented in the CWI/Oregon software is the algorithm by Montgomery as explained in Section 3.1.9. The implementation makes use of the PARI software package [2] for algebraic manipulations and extraction of the square root when the coefficients have been reduced to a reasonable size. The output from the square root algorithm is the factors of N.

4.3 Choice of the sieving area

The sieving area is the range of a- and b-values, $a, b \in \mathbb{Z}, b > 0$, from which we try to find the relations. When speaking of the sieving area, a real root r of a polynomial f is the line $r = a/b$, extending from the origin, such that $f(r) = 0$. See Figures 4.1 and 4.2 for an example of a plot of a and b.

The choice of the sieving area will affect the size of the norms from both polynomials. As explained in Section 4.2.1, we have one linear polynomial and one of higher degree. Large values of a and b will not affect the norm from the linear polynomial as much as the norm from the higher degree polynomial. Thus, we will concentrate on finding a good sieving area for the higher degree polynomial.

If the sieving area is chosen properly the size of the norm is small, giving us a higher probability that the norms are smooth and the running times decrease. Experiments based on the ideas given below are found in Section 4.5. The main ideas were:

a) Sieve a polygon around the origin to cover the ground where the real roots come together. If necessary, sieve a little wider for small b-values since there are many relations there because of the small norm.

It may be difficult to determine the size and shape of the sieving area. Perhaps this could first be sampled with the line sampler to get a picture of how the norms decrease.

b) Sieve on the real roots of the polynomials for as many b-values as profitable.

The profit would of course depend on the slope on the side of the ridges. As for any differentiable function, the values vary with the derivatives.

c) Within a certain area, let the length of the a-interval depend on the gcd(b, *leading coefficient of the polynomial*) so that some promising b-values are more thoroughly investigated.

The idea here is to decrease the size of the norm by applying (4.3). If the remainder is of reasonable size, there is a much greater chance that it is B-smooth.

4.4 Implementation of a sampler

To minimize the running times needed, we implemented a parallel sampler that uses MPI (message passing interface) [9], a parallel software package, to run on several computers.

We wrote three programs to investigate the ideas about the sieving area in the previous section. First, we needed a program that sampled a rectangular sieving area. Second, we were interested in how the norm behaved near the real roots of the polynomials. Third, we were interested in how the gcd(b, *leading coefficient of the polynomial*) affected the relation count.

The samplers were then used for the experiments in the next section.

4.5 Experiments

Most of the following experiments use the sieve together with the samplers discussed in the previous section to answer some questions about the sieving area and the norm. Throughout the sieve experiments we have used a 101 digit number

$$\begin{aligned} n &= 1367422137\ 6761637153\ 8058299847\ 0528750457 \\ &\quad 7638778523\ 0230764380\ 1173554513\ 1921835577 \\ &\quad 8851619009\ 1643684532\ 9 \end{aligned}$$

with polynomials

$$\begin{aligned} f_1 &= -39601549612794148559671 + x \\ f_2 &= 139543380039440821338034 80 + \\ &\quad 4906668047110889964262\ x + \\ &\quad -587274291267665623\ x^2 + \\ &\quad -23147725526270\ x^3 + \\ &\quad 5559732360\ x^4. \end{aligned}$$

The factors of the leading coefficient of the fourth degree polynomial are $5559732360 = 2^3 \cdot 3^2 \cdot 5 \cdot 7 \cdot 13 \cdot 17 \cdot 67 \cdot 149$. The factors of the second highest coefficient are $23147725526270 = 2 \cdot 5 \cdot 17 \cdot 5119 \cdot 26599549$.

4.5.1 Sieve prerequisites

There are a few things to determine before starting the sieve. We must choose the factor base bounds B_j and L_j for both polynomials. These are best determined experimentally, and a few examples of these bounds are listed in the CWI/Oregon article [7] on which we based our choices for the numbers we experimented with.

One must also choose an approximation of how many relations that should be collected. As stated in [7], one approximation formula used is

$$no\ of\ relations = 0.8(\pi(L_1) + \pi(L_2) - \pi(\min(L_1, L_2))/g),$$

where g is the product of the order of the Galois groups of the polynomials, as discussed in Section 3.1.8.

4.5.2 Sieving a rectangle

In this section, we describe how we sieved to find all relations in a large rectangle.

In the factorization of the 101 digit number, we chose $L_1 = L_2 = 10^7$ and $B_1 = B_2 = 1.2 \cdot 10^6$. Then

$$\begin{aligned} no\ of\ relations &= 0.8(\pi(L_1) + \pi(L_2) - \pi(\min(L_1, L_2))/g) \\ &= 0.8(664579 + 664579 + 664579/4!) \approx 1.05 \cdot 10^6. \end{aligned}$$

The roots of the fourth degree polynomial were -2299.934432 and -10943.30080. The norm has great probability of being small on the real root of a polynomial as explained in Section 4.3 b), and thus we wanted the smaller root to fit within the sieving area. The larger root was considered too big to fit within the sieving area because of the growth of the a-values for big b-values. We chose the sieving area to $b \leq 50000, |a| \leq 1 \cdot 10^8$. The sieving area is shown in Figures 4.1 and 4.2.

When the sieving was completed we had gathered about $1.05 \cdot 10^6$ relations. The total running time was $3.9 \cdot 10^6$ seconds or 46 CPU days on about 25 Sparc 5 Workstations, 333 MHz. Wall clock time was almost two days.

From observations of Figure 4.2, we could see that the sieving area chosen above had quite a large norm in one of the corners. From the figure we deduced that a better rectangle would be $b \in [1, 18000]$ and $|a| \leq 1.8 \cdot 10^8$ since the norm would then be bounded from above by about 42 digits instead of 44. When sieving this area, all relations were gathered in $2.5 \cdot 10^6$ seconds or 29.0 CPU days on the same computers as above, which is much faster than the first rectangle. Wall clock time was about one and a half day.

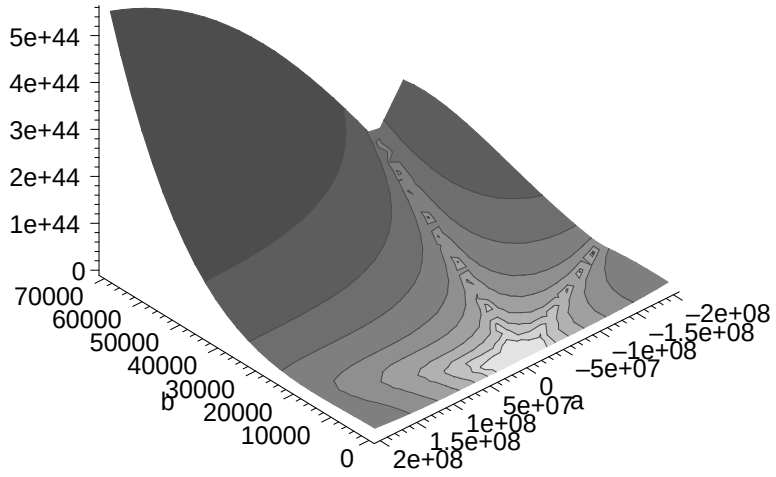

Figure 4.1. *Norm of the fourth degree polynomial. In the lightest area around the origin, the norm is below* $1 \cdot 10^{40}$ *and with every other curve the norm increases with one power of 10. The two valleys are the real roots of the polynomial.*

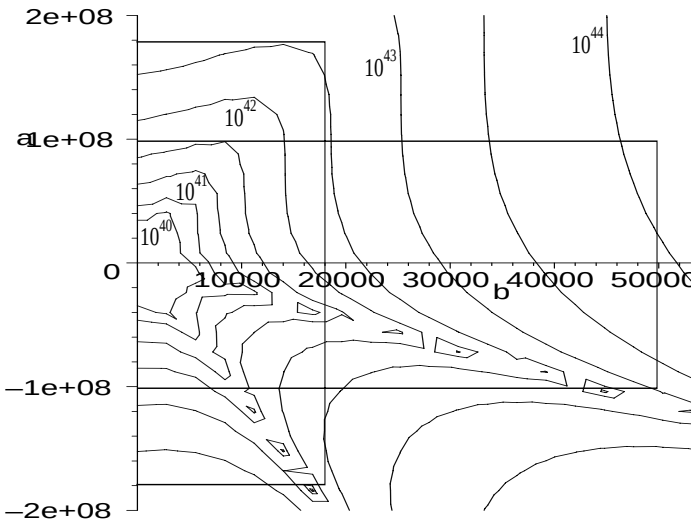

Figure 4.2. *The sieving area from Figure 4.1 seen from above. The curves correspond to a constant level of the norm. The sieving area chosen for the 101 digit experimental number was* $b \in [1, 50000], |a| \leq 1 \cdot 10^8$. *We see that this region cuts deep into areas with a large norm which most probably yield fewer relations than corresponding areas with small norm. A sieving area with smaller upper bound on the norm is* $b \in [1, 18000], |a| \leq 1.8 \cdot 10^8$.

4.5.3 Implications of the greatest common divisor

In this section, we investigate how the $\gcd(b, c_d)$, where c_d is the leading coefficient of the polynomial, affects the number of relations found. We know from equation (4.3) that $\gcd(b, c_d)$ is a factor of the norm and therefore, the remainder will be small. We also investigate how a and b, that are not coprime, affect the speed of the sieve.

We investigated how a small prime divisor p of b actually decreased the relation count. Given a p, we estimated the number of potential relations by counting the number of a- and b-values with $\gcd(a, b) \neq 1$. We were also interested in how the $\gcd(b, c_d)$ affected the relation count. For each of the values p of interest, we counted the number of relations found when $\gcd(b, c_d) = p$ and $\gcd(b, c_d) < p$. In Table 4.1 we give the results for different values of p. For our fourth degree polynomial, we have that 2, 3, 5, 7, 13, 17, 67 and 149 are divisors of the leading coefficient c_d. The other values are prime numbers close to the divisors of c_d.

From the values in Table 4.1, we see that if $p = \gcd(b, c_d)$ then the yield was greater than one. This implies that it was more profitable to sieve values of b such that $p|b$ than those with $p \not| b$, if we could sieve the same amount of

potential relations independently of p. Unfortunately, the sieve also spends time on the relations with $\gcd(a, b) \neq 1$.

When using the sieve, we wanted to find the maximum amount of relations per time unit. Even though we had a greater yield for small primes p, the time spent on the relations that had $\gcd(a, b) \neq 1$ gave us a lower relation count per time unit. Thus, we wanted to maximize the potential relations searched per time unit, taking the yield and speed of the sieve into account. To test these ideas, we used an ad hoc formula that did not take into account that the norms increase non-linearly. We set the length of the a-interval for a given b to

$$\text{length}_b = \text{length} \cdot \frac{\text{potential relations for } p|b}{\text{potential relations for } p \not| b} \cdot \text{yield} \cdot \text{sieve speed}, \qquad (4.4)$$

where p is the smallest divisor of b. The speed of the sieve is about the same for all $p|b$ except for $p = 2$, where the speed of the sieve is about 1.8 times faster because of an implementational feature. The values are given in Table 4.1.

To compare with the sieving of a rectangular area in the previous section, we sieved a similar area using variable lengths of the a-intervals. We chose $b \leq 18000$ and used $|a| \leq 1.8 \cdot 10^8$ as the original length of the a-interval. When all b-values had been sieved, we had collected 885447 relations in $2.2 \cdot 10^6$ CPU seconds, or 25.9 CPU days, which is 12% faster than the rectangle sieved but with 16% fewer relations found. Thus, with these polynomials, the rectangular area is better.

4.5.4 Sieving a contour

A small norm has greater probability of being smooth than a large one. Therefore, in theory, a rectangular sieving is not optimal if we want to include the roots in the sieving area. In practice, the sieve is optimized for large a-intervals which can decrease the running times when sieving a rectangle. In this section we present the experiments we did with a non-rectangular sieving area.

By observing Figure 4.3, we see that the norm can be bounded to about 42.5 decimal digits by using the contour indicated. We sieved this area and collected 959958 relations in $28.7 \cdot 10^6$ CPU seconds, or 33.2 CPU days, to be compared with the 29 CPU days for the rectangle sieved. For this example number and polynomial, we had no increase in speed. This is because of the rectangular shape of the contours near the origin where almost all relations needed can be found. In the experiment we found only 10% of the relations when sieving the triangles covering the roots of the polynomial.

p	$p\|b$	$p\not\|b$	*Sieve*	*Yield*	*Interval*
2	0.385	0.772	1.8	1.095	0.983
3	0.400	0.600	1.0	1.011	0.674
5	0.432	0.540	1.0	1.107	0.886
7	0.452	0.527	1.0	1.207	1.035
11	0.472	0.519	1.0	0.871	0.792
17	0.486	0.516	1.0	1.540	1.450
19	0.498	0.516	1.0	0.898	0.852
67	0.509	0.515	1.0	2.167	2.142
71	0.509	0.515	1.0	0.907	0.898
149	0.514	0.515	1.0	2.562	2.561
151	0.514	0.515	1.0	0.999	0.999
997	0.529	0.515	1.0	0.891	0.916

Table 4.1. *Experiments with the greatest common divisor. The second column is the density of pairs (a, b) for which $gcd(a, b) = 1$ with $p|b$. The third column is the density for b with $p\not|b$. The fourth is the estimated speed of the sieve when $p|b$. The fifth is the density of relations when the same number of pairs (a, b) with $gcd(a, b) = 1$ is sieved. The last column is the length of the a-interval sieved to obtain the maximum number of potential relations searched per time unit.*

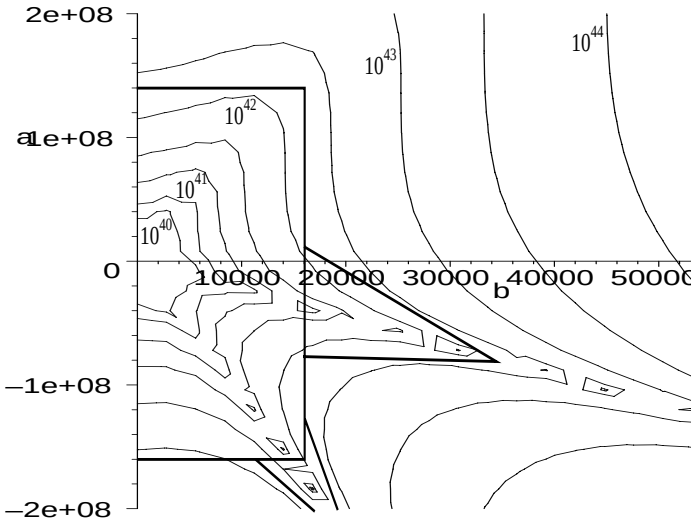

Figure 4.3. *Sieving a contour. The sieving area is chosen so that the largest norm is as small as possible while finding the relations needed.*

4.5.5 Smoothness of the norm

The smoothness of the norm determines how fast the relations can be gathered and therefore, the speed of the algorithm. We wanted to examine the smoothness of the norm as compared to the smoothness of an arbitrary number. Another interesting aspect was how smooth the norm was after applying equation (4.3).

For the random numbers, we chose five arbitrary numbers and checked the consecutive million numbers for smoothness. For the norm, we chose five consecutive b-values and then sieved one million a-values for each. The experiment with the reduced norms was much like the norm experiment except that we used three hand picked b-values.

The results showed that the norm was several times smoother than an arbitrary number, and so also in the case with the reduced norms. All values are given in Table 4.2. Remember here that the smoothness of the norm from one polynomial does not suffice to form a relation since it is necessary for both norms to be smooth.

The additional smoothness of the norm over an arbitrary number is an interesting property of the polynomial. We know from equation (4.1) that if $a = br \bmod p$ then p divides $F(a, b)$, where r is a root to $f(r) = 0 \bmod p$. Each prime $p \leq B_j$ gives us a stochastic variable $X_p = \log(p)$ with probability c_p/p, where c_p is the number of roots for p. With $c_p = 1$ we have a stochastic variable for a random number. If we introduce $X = \sum_p X_p$, we have that X of the digits of a large number is B_j-smooth. The expected value is

$$E(X) = E\Big(\sum_{p \leq B_j} X_p\Big) = \sum_{p \leq B_j} E(X_p) = \sum_{p \leq B_j} \frac{c_p \cdot \log(p)}{p},$$

where all p are prime.

With the norms from the fourth degree example polynomial, we had $E(X) = 7.365$. For the random numbers (that is, $c_p = 1$), we got $E(X') = 5.518$. The standard deviation was $D(X) = 10.094$ and $D(X') = 9.774$, respectively. We assumed that the probability of a number being B_j-smooth followed a normal distribution $N(E(X), D(X))$ and $N(E(X'), D(X'))$, respectively. We wanted to compare this approximation with the experimental values to get an explanation of the big difference in the smoothness between a random number and the norm.

To give an example, we use the 36 digit decimal numbers. We used the bounds $B_j = 1.25 \cdot 10^6$ and $L_j = 10^7$ from the previous experiments. We wanted the numbers to have a remainder below L_j^2 when all primes below B_j were divided out. Actually, this is not the same as a number being B_j-smooth with at most two large primes, and we dealt with this as explained below.

log_{10}	D_{B_j}	D_{L_j}	$D_{L_j^2}$	*Norm*	*Predicted*	*Reduced*
30	0.20%	0.22%	4.69%	-	0.48%	-
31	0.13%	0.15%	3.51%	-	0.33%	-
32	0.090%	0.10%	2.60%	-	0.23%	-
33	0.060%	0.070%	1.92%	-	0.16%	-
34	0.038%	0.046%	1.39%	-	0.10%	-
35	0.026%	0.033%	1.00%	-	0.073%	-
36	0.016%	0.020%	0.72%	0.12%	0.045%	0.10%
37	0.012%	0.014%	0.52%	0.11%	0.033%	0.078%
38	0.0079%	0.010%	0.36%	0.082%	0.022%	0.050%
39	0.0040%	0.0057%	0.25%	0.044%	0.010%	0.039%
40	0.0031%	0.0047%	0.17%	0.028%	0.0082%	0.023%
41	0.0014%	0.0024%	0.12%	0.018%	0.0035%	0.019%
42	0.0013%	0.0016%	0.084%	0.016%	0.0030%	0.014%
43	0.0010%	0.0010%	0.054%	0.010%	0.0023%	0.0085%
44	0.0006%	0.0009%	0.041%	0.0079%	0.0011%	0.0051%

Table 4.2. *Smoothness of the norm and random numbers. The first column is the number of decimal digits. The next three columns are experimental values of the density of B_j-smooth numbers with at most two large primes $\leq L_j$, L_j-smooth numbers and numbers with remainder $\leq L_j^2$ after dividing out primes $\leq B_j$, respectively. The fifth column is how many of the norms from the fourth degree polynomial that are smooth and the sixth is the predicted density of relations. The last column is how many smooth numbers there are when the norm has been reduced using equation (4.3). The sizes of the last column are after the reduction explained in the text.*

When evaluating the density of the normal distributions between $36 - 2 \cdot \log_{10}(L_j) = 22$ and 36 we got $I = 0.02013$ and $I' = 0.00854$. From Table 4.2 we have that the density of B_j-smooth numbers with at most two large primes was $D_{B_j} = 0.00016$ while the density of numbers with remainder L_j^2 after having divided out all primes below B_j was $D_{L_j^2} = 0.0072$. The latter corresponds well with $I' = 0.00854$ as predicted. From this, we estimated the number of norms that are B_j-smooth with at most two large primes to be

$$I \cdot \frac{D_{B_j}}{D_{L_j^2}} = 0.02013 \cdot \frac{0.00016}{0.0072} = 0.00045,$$

which should be compared to the value 0.0012 from Table 4.2. The difference could perhaps be due to the additional smoothness of the norm from equation (4.3).

4.6 Implementation of a sieve

To get a better understanding of how the sieve works, an implementation was made as shown in algorithm 1. The implementation was of mere theoretical interest and not used in any of the experiments.

The implementation is quite straightforward and uses all the details discussed in Section 4.2.3. The array is initialized at line (3) with the logarithm of the gcd(b, *leading coefficient of polynomial*) subtracted from the norm, making use of equation (4.3). Line (8) finds the first occurrence of a satisfying $a = br \bmod p$ from equation (4.1) and then $\log(p)$ is subtracted from every pth element of the array in lines (9)–(11).

When log of all factors have been subtracted, the array elements are checked for potential smoothness at line (13) using the threshold from Section 4.2.3. The factors are then found at lines (14) and (17) using the same techniques as in lines (3) and (8). If all factors are below the factor base bound for large primes L_j for both of the polynomials, the relation is approved and is sent to output.

4.7 Conclusions

4.7.1 The sieving area

The experiments dismissed most of the suggestions made in Section 4.3, and it appears as if a rectangle would be a good basis for the sieving area. The experiments with the contours from Section 4.5.4 gave no improvement in speed over the rectangular shape. From the experiments with the greatest common divisor, we had the same result.

The results from the three experiments had almost the same outcome measured in relations per time unit. With a polynomial having a contour that is not as easy to approximate with a rectangle, the performance of the contour experiment might be better.

When observing Figure 4.2, we see that the norm increases with a large power of a. The approximative formula (4.4) does not take such growth into account. A more elaborate formula should use this knowledge to find an interval for each b-value so that the largest norm is kept as small as possible.

4.7.2 Smoothness of the norm

In Section 4.5.5 we used a normal distribution to find an approximation of the smoothness of the norm. The approximation was good for the random numbers and gave a low estimate of the density of relations for different sizes of the norm.

Algorithm 1: An implementation of the sieve of the NFS
Input: The b-value to be sieved, upper and lower bounds for the a-interval, two polynomials p_1 and p_2 and the factor base bounds B_j and L_j for both polynomials.
Output: Relations (a, b) where the norms from both polynomials are smooth.

SIEVE$(b, a_{min}, a_{max}, p_1, p_2, B_1, B_2, L_1, L_2)$
(1) **foreach** $a = a_{min} \ldots a_{max}$
(2) **foreach** $i = 1 \ldots 2$
(3) $array_i[a - a_{min}] \leftarrow$ LOG(NORM(a, b, p_i) - GCD(b, leading coefficient of p_i))
(4) **foreach** $a = a_{min} \ldots a_{max}$
(5) **foreach** $i = 1 \ldots 2$
(6) **foreach** prime number $p \leq B_i$
(7) **foreach** solution r of $p_i(r) = 0 \bmod p$
(8) $a' \leftarrow$ FIRST-A-EQUAL-BR(a_{min}, p)
(9) **while** $a' \leq a_{max}$
(10) $array_i[a' - a_{min}] \leftarrow array_i[a' - a_{min}]$ - LOG(p)
(11) $a' \leftarrow a' + p$
(12) **foreach** $a = a_{min} \ldots a_{max}$
(13) **if** $array_1[a - a_{min}]$ below threshold from L_1 **and** $array_2[a - a_{min}]$ below threshold from L_2
(14) $factors_1 \leftarrow$ FIND-FACTORS(a, b, p_1)
(15) **if** found factor above L_1 among $factors_1$
(16) **continue**
(17) $factors_2 \leftarrow$ FIND-FACTORS(a, b, p_2)
(18) **if** found factor above L_2 among $factors_2$
(19) **continue**
(20) OUTPUT-RELATION$(a, b, factors_1, factors_2)$

Appendix A

Dictionary

For any concept not covered here, we refer to [5] and [10].

Algebraic number A real number that is a root to some polynomial with coefficients in $\mathbb{Q}$. All numbers that are not algebraic are called transcendental.

Algebraic number field Given a polynomial of degree d with a root α, we get an algebraic number field $\mathbb{Q}(\alpha)$ which is the smallest field containing both $\mathbb{Q}$ and α. The numbers in an algebraic number field can be seen as polynomials in α and coefficients in $\mathbb{Q}$. For example, $5 - \frac{7}{2}\alpha + 2\alpha^2$, and d is at least 3.

Embedding An embedding is a homomorphism from $\mathbb{Q}(\alpha)$ to $\mathbb{C}$, mapping each root to a given root. For example, if we have $x^2 - 2 = 0$ with roots $\pm\sqrt{2}$ and an embedding $\theta : x \to \sqrt{-2}$ we have that $\theta(3+4\sqrt{2}) = 3 - 4\sqrt{2}$. A polynomial with n roots have one embedding per root, numbered $\theta_1, \theta_2, \ldots, \theta_n$.

Field A ring where all nonzero elements have a multiplicative inverse. For example, the rational numbers $\mathbb{Q}$.

Homomorphism A map $\phi : R \to R$ that satisfies $\phi(ab) = \phi(a)\phi(b)$ and $\phi(a + b) = \phi(a) + \phi(b)$ is called a ring homomorphism on the ring R.

Ideals An ideal $\mathcal{I}$ is a subset of a ring R with the property that $a\mathcal{I} \in \mathcal{I}$ and $\mathcal{I}a \in \mathcal{I}$ for all $a \in R$. That is, an ideal is closed under multiplication and addition with all elements of the ring.

Irreducible element Element that is not divisible by any other element than a unit and itself.

Number ring Given a monic polynomial with root α, a number ring over $\mathbb{Z}$ is a subset of an algebraic number field $\mathbb{Q}(\alpha)$ that forms a ring, written

$\mathbb{Z}[\alpha]$. The subset is a ring rather than a field since not all elements have to have an inverse.

Ring A set of elements associated with two operations, normally called addition and multiplication. The operations have to satisfy a set of conditions (not given here). One example of a ring is the integers $\mathbb{Z}$.

Smooth A number is called k-smooth if all its prime factors are less than k. For example, $15 = 5 \cdot 3$ is 5-smooth while $14 = 7 \cdot 4$ is not. An algebraic number is called k-smooth if its norm is k-smooth.

Bibliography

1. L. Adleman, R. L. Rivest, and A. Shamir. A method for obtaining digital signature and public-key cryptosystems. *Communication of the ACM*, 21(2), 1978.

2. C. Batut, D. Bernardi, H. Cohen, and M. Olivier. *User's Guide to PARI-GP*. Distributed with the system from anonymous FTP at `ftp://megrez.math.u-bordeaux.fr/pub/pari/`, last visited in June, 2000.

3. D. Coppersmith. Solving linear equations over GF(2): Block lanczos algorithm. *Linear algebra and its applications*, (192):33–60, 1993.

4. Jean-Marc Couveignes. Computing a square root for the number field sieve. *Lecture notes in mathematics*, 1(1554):95–102, 1993.

5. John B. Fraleigh. *Abstract algebra*. Addison-Wesley, 1999.

6. David Hawkins. Mathematical sieves. *Scientific American*, 199(6):105–??, December 1958.

7. R. Marije Huizing. An implementation of the number field sieve. Technical report, CWI, 1995.

8. A. K. Lenstra, H. W. Lenstra Jr., M. S. Manasse, and J. M. Pollard. The number field sieve. *Lecture notes in mathematics*, 1(1554):11–42, 1993.

9. ANL Mathematics and Computer Science. *User's Guide to MPI*. Home page at `http://www.msc.anl.gov/mpi/index.html`, last visited in June, 2000.

10. Rickard A. Mollin. *Algebraic number theory*. CRC Press LLC, 1999.

11. J. M. Pollard. A monte carlo method for factorization. *BIT*, 3(15):331–334, 1975.

12. Carl Pomerance. The quadratic sieve factoring algorithm. In T. Beth, N. Cot, and I. Ingemarsson, editors, *Advances in Cryptology: Proceedings of EUROCRYPT 84*, volume 209 of *Lecture Notes in Computer Science*, pages 169–182. Springer-Verlag, 1985, 9–11 April 1984.

13. Ian Stewart. *Galois theory*. Chapman and Hall, Ltd, 1973.