Faktorisieren mit dem Number Field Sieve

Dissertation
zur Erlangung des Grades
des Doktors der Ingenieurwissenschaften
der Technischen Fakultät
der Universität des Saarlandes
von

Jörg Zayer

Saarbrücken
1995

Hiermit versichere ich an Eides Statt, daß ich diese Arbeit selbständig verfaßt und keine anderen als die angegebenen Quellen benutzt habe.

Saarbrücken, 20.Juni 1995

Mein besonderer Dank gilt Herrn Prof. Dr. J. Buchmann, der es mir ermöglichte, an diesem interessanten Thema zu arbeiten.
Ferner danke ich Thomas Denny, Jürgen Loho und Damian Weber für die zahlreichen Diskussionen und die vielen hilfreichen Anregungen.
Thomas Papanikolaou danke ich für die großartige Unterstützung in meinem Kampf mit C, C++ und LaTeX.
Abschließend möchte ich Karen Blaich, Klaus Kiefer, Thomas Sosnowski und Damian Weber für das Korrekturlesen danken.

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Einleitung

Gegenstand dieser Arbeit ist das Faktorisieren großer natürlicher Zahlen mit dem *Number Field Sieve* Algorithmus (NFS).
Das Zerlegen natürlicher Zahlen in ihre Primfaktoren ist ein Problem, mit dem sich schon die Mathematiker früherer Jahrhunderte, wie z.B. Gauss oder Fermat, beschäftigt haben. War es bei diesen Wissenschaftlern noch reines Interesse an der Mathematik, so hat das Faktorisieren natürlicher Zahlen in der heutigen Zeit auch Anwendungen gefunden. Beispielsweise werden in der Kryptographie, u.a. im RSA–Verschlüsselungsalgorithmus, große, zusammengesetzte, natürliche Zahlen verwendet, um Nachrichten zu kodieren. Die Sicherheit dieses Verfahrens hängt von der Schwierigkeit ab, die Primteiler dieser Zahlen zu bestimmen.
Seitdem Computer zum Faktorisieren eingesetzt werden können, wurden immer wieder neue Algorithmen in diesem Bereich entworfen.
Der neueste ist der NFS Algorithmus. Dieser hat sich aus einer Arbeit von J. M. Pollard aus dem Jahre 1988 ([Po88]) entwickelt. Pollard beschreibt dort eine Methode zum Faktorisieren von speziellen natürlichen Zahlen. Diese müssen im Bildbereich des Polynoms $f(x) = x^3 + 2$ liegen. Pollards Verfahren arbeitet einerseits im Ring der ganzen Zahlen $\mathbb{Z}$ sowie andererseits im Ring der ganzen algebraischen Zahlen des Zahlkörpers $\mathbb{Q}(\sqrt[3]{-2})$. Mit diesem Algorithmus war er in der Lage, die siebte Fermatzahl (39 Dezimalstellen) zu faktorisieren.
Aus dieser Arbeit ging der spezielle *Number Field Sieve* Algorithmus ([LLMP90] oder [Za91]) hervor, der es ermöglicht, zusammengesetzte Zahlen zu zerlegen, die im Bildbereich normierter Polynome mit einstelligen Koeffizienten liegen. Mit diesem Verfahren gelang es 1990, die 155–stellige neunte Fermatzahl zu faktorisieren.
Seit diesem Zeitpunkt wurde mit Nachdruck daran gearbeitet, diesen Algorithmus für zusammengesetzte Zahlen ohne spezielle Eigenschaften nutzbar zu machen. Im Spätfrühjahr des Jahres 1992 veröffentlichten J. P. Buhler, H. W. Lenstra Jr. und C. Pomerance eine Arbeit ([BuLePo93]), in der sie ein solches Verfahren beschrieben. Sie konnten durch eine heuristische Laufzeitanalyse zeigen, daß das *Number Field Sieve* der asymptotisch schnellste, bekannte Faktorisierungsalgorithmus für beliebige Zahlen ist. Allerdings war ihre Arbeit rein theoretisch.
Es war zu diesem Zeitpunkt keineswegs klar, ob dieses Verfahren überhaupt prak-

tikabel ist. Zwar konnte gezeigt werden, daß für den allgemeinen Fall in etwa die gleichen Siebprozeduren (siehe Kapitel 2.3.1) verwendet werden konnten wie im speziellen *Number Field Sieve* Algorithmus, allerdings konnte weder dem erhöhten Bedarf an Relationen in effizienter Weise Rechnung getragen, noch ein praktikabler Algorithmus zur Berechnung der Quadratwurzel aus dem algebraischen Quadrat (siehe Kapitel 2.4) angegeben werden. Erst durch die Beobachtungen von J. - M. Couveignes ([Cv94]) konnte letzteres Problem auch für die Praxis gelöst werden.
Die ersten Implementierungen des NFS entstanden am Bellcore–Institut in den USA ([BeLe93]) und an der Universität des Saarlandes ([BuLoZa93]). Während die Amerikaner ihre Implementierung nutzten, um Zahlen spezieller Form mit etwa 150 Dezimalstellen zu zerlegen, wurde im Januar 1993 in Saarbrücken zum ersten Mal eine 29–stellige allgemeine Zahl, d.h. ohne spezielle Eigenschaften, mit dem NFS Algorithmus faktorisiert.
In dieser Arbeit werde ich neben einer kurzen Erläuterung des *Number Field Sieve* Algorithmus sowie einiger, schon bekannter Varianten, auch neue effizientere Algorithmen beschreiben. In diesem Zusammenhang sind vor allem die Quadruple Large Prime Variante, die eine Verbesserung der Relationenberechnung darstellt, sowie die Beschleunigung von Couveignes Algorithmus zur Berechnung der Quadratwurzel durch Reduktion der Quadrate zu nennen. Auch werde ich mich ausführlich mit der Berechnung des Polynoms, das den zugrundeliegenden Zahlkörper definiert, beschäftigen und eine neue Methode angeben, mit der man, ohne die Faktorisierung explizit durchzuführen, vorhersagen kann, mit welchem von mehreren gegebenen Polynomen die Faktorisierung am schnellsten zu realisieren ist.

1.2 Gliederung der Arbeit

Im weiteren Verlauf dieses Kapitels werde ich die verwendete Darstellung der Algorithmen beschreiben. Anschließend werden mathematische Grundlagen zusammengestellt, die für das Verständnis des *Number Field Sieve* Algorithmus benötigt werden.

In Kapitel 2 wird die Grundidee des NFS Algorithmus beschrieben und das Verfahren anhand eines kleinen Beispiels durchgeführt.

Kapitel 3 widmet sich dem Herz des NFS, der Siebphase. Hier werden zuerst einige schon aus anderen Faktorisierungsverfahren bekannte Verbesserungen beschrieben. Anschließend wird die Idee einer neuen Methode, der Quadruple Large Prime Variante, vermittelt, die dazu benötigten Algorithmen beschrieben und die Effizienz dieser Variante untersucht. Danach wird auf eine Modifikation der Siebphase, das von J. Pollard eingeführte Lattice Sieve, eingegangen. Den Abschluß dieses Kapitels bildet ein Abschnitt über die Möglichkeit die Relationen auf mehreren Rechnern verteilt zu generieren.

Kapitel 4 beschäftigt sich mit der Wahl des Polynoms und somit mit der Wahl des Zahlkörpers, mit dem im *Number Field Sieve* gearbeitet wird. Dort werde ich außerdem eine neue Methode angegeben, die es in kurzer Zeit ermöglicht zu entscheiden, welches von mehreren gegebenen Polynomen zur schnellsten Faktorisierung führt.

Des Weiteren werde ich ein probabilistisches Verfahren zur Generierung gut geeigneter Polynome angegeben.

In Kapitel 5 wird der letzte Schritt des NFS Algorithmus diskutiert. Vor allem wird dort eine neue, effiziente Variante zum Berechnen der Quadratwurzel eines algebraischen Zahlquadrates beschrieben. Abschließend wird noch die Frage erörtert, wie zufällig die vom NFS erzeugten Lösungen der quadratischen Kongruenz in der Praxis sind.

In Kapitel 6 werden die Auswirkungen verschiedener Parameter auf die Laufzeit des *Number Field Sieve* in der Praxis besprochen und eine Tabelle geeigneter Parameter aufgeführt.

Kapitel 7 enthält eine Reihe von Steckbriefen von Faktorisierungen, die ich unter Verwendung meiner Implementierung des NFS durchgeführt habe.

Im Anhang sind einige fundamentale zahlentheoretische Algorithmen beschrieben, die im NFS Algorithmus an der einen oder anderen Stelle benötigt werden. Des Weiteren findet sich dort eine Liste der in dieser Arbeit beschriebenen Algorithmen, sowie eine Liste der Abbildungen und Tabellen.

1.3 Darstellung der Algorithmen

Zur Darstellung der Algorithmen verwende ich in dieser Arbeit den von Thomas Papanikolaou und mir entwickelten TeX Algorithmenstil.
Ein Algorithmus besteht danach aus zwei Teilen: dem Kopf- und dem Anweisungsteil. Der Kopf eines Algorithmus enthält den Namen des Algorithmus und definiert Ein- und Ausgabe:

Algorithmenname

EINGABE: Eingabeparameter
AUSGABE: Ausgabeparameter

Eine Anweisung besteht aus einer Zeilennummer, einem oder mehreren Befehlen und einem Kommentar, der durch die Kommentarklammern /* */ der Programmiersprache C gekennzeichnet ist:

(5) Anweisung /* Kommentar */

Ein Schleifenrumpf beginnt mit dem Schlüsselwort **do** und wird mit dem Schlüsselwort **od** abgeschlossen. Wir unterscheiden dabei Zählschleifen:

(2) **for** (Bedingung) **do** /* Kommentar */
(3) Anweisungsblock
(4) **od**

oder Schleifen, deren Fortsetzung von der Erfüllung einer Bedingung abhängig ist:

(2) **while** (Bedingung) **do** /* Kommentar */
(3) Anweisungsblock
(4) **od**

bzw.

(3) **repeat**
(4) Anweisungsblock
(5) **until** (Bedingung) /* Kommentar */

Soll eine Schleife für Elemente einer Menge ausgeführt werden, so ist dies auch durch die **foreach**–Schleife möglich.

(3) **foreach** (Element der Menge) **do** /* Kommentar */
(4) Anweisungsblock
(5) **od**

Verzweigungen werden durch den **if ... then ... else ... fi** Konstrukt dargestellt:

(3) **if** (Bedingung) **then** /* Kommentar */
(4) Anweisungsblock1
(5) **else**
(6) Anweisungsblock2
(7) **fi**

Ist der Anweisungsblock2 leer, so wird der **else**-Fall weggelassen. Ebenso kann das **fi** entfallen, wenn die Anweisung im **then**-Teil nur aus einer Zeile besteht.
Neben den hier beschriebenen Konstrukten verwende ich auch nichtnumerierte unterstrichene Kommentarzeilen, um die Gliederung des Algorithmus deutlich hervorzuheben.

ERSTER TEIL

(3) Anweisungsblock1

ZWEITER TEIL

(4) Anweisungsblock2

Die Rückgabe von einem oder mehreren Werten geschieht durch

(3) **return** (wert1, wert2, ...) /* Kommentar */

1.4 Mathematische Grundlagen

In diesem Abschnitt stelle ich die zum Verständnis des *Number Field Sieve* notwendigen Begriffe zusammen. Diese stammen größtenteils aus der elementaren und algebraischen Zahlentheorie. Als Quellen dienten hierzu in erster Linie die Arbeiten von Stewart und Tall ([StTa87]) sowie von Hecke ([Hec]). Aussagen, die anderen Vorlagen entnommen sind, sind gesondert gekennzeichnet.

1.4.1 Bezeichnungen

An dieser Stelle möchte ich Bezeichnungen einführen, die im weiteren Verlauf der Arbeit benutzt werden, dort aber nicht mehr explizit definiert werden.

$\mathbb{N}$	die Menge der natürlichen Zahlen
$\mathbb{N}_0$	$\mathbb{N} \cup \{0\}$
$\mathbb{P}$	die Menge der Primzahlen
$\mathbb{Z}$	der Ring der ganzen Zahlen
$\mathbb{Z}/n\mathbb{Z}$	der Restklassenring modulo n
$\mathbb{Z}/n\mathbb{Z}^*$	die multiplikative Gruppe des Restklassenringes modulo n
$\mathbb{Q}$	der Körper der rationalen Zahlen
$\mathbb{R}$	der Körper der reellen Zahlen
$\mathbb{C}$	der Körper der komplexen Zahlen
$\mathbb{F}_{p^d}$	der Körper mit p^d Elementen
$M_{>m}$	die Teilmenge einer geordneten Menge M mit den Elementen größer m

Für eine endliche Menge M bezeichnet $|M|$ die Anzahl der Elemente in M, also die Ordnung von M.

Für eine Primzahl $p \in \mathbb{P}$ und eine ganze Zahl $z \in \mathbb{Z}$ sei $\mathrm{ord}_p(z)$ der Exponent von p in der Primfaktorzerlegung von z, z.B. ist $\mathrm{ord}_3(63) = \mathrm{ord}_3(3^2 \cdot 7) = 2$.

Für $r \in \mathbb{R}$ bezeichne ich mit $\lfloor r \rfloor$ die größte ganze Zahl z mit $z \leq r$, mit $\lceil r \rceil$ die kleinste ganze Zahl z mit $z \geq r$ und mit $[r]$ die zu r nächstgelegene ganze Zahl.

Für einen Vektor $\vec{v} \in \mathbb{Z}^n$ bezeichnet v_i die i-te Komponente.

Mit Z_i bzw. P_i ist eine i–stellige zusammengesezte Zahl bzw. Primzahl gemeint;

n steht in der gesamten Arbeit für die zu faktorisierende Zahl.

1.4.2 Zahlkörper

Definition 1.1 (Polynomring, Polynome, Wurzel eines Polynoms)

1. *Sei K ein Körper, x eine Unbestimmte über K, so nennt man*

$$K[x] := \{f_0 + f_1 \cdot x + \ldots + f_d \cdot x^d \mid d \in \mathbb{N}_0, f_i \in K\}$$

   *den **Polynomring** über K.*

2. *Ein Element $f(x) = f_0 + f_1 \cdot x + \ldots + f_d \cdot x^d \in K[x]$ mit $f_d \neq 0$ heißt **Polynom** vom **Grad** d.*

3. *Ein Element $\rho \in K$ mit $f(\rho) = 0$ heißt **Wurzel** des Polynoms $f(x)$.*

Definition 1.2 (Zahlkörper)

Sei $\mathbb{K}$ eine mindestens zweielementige Teilmenge der komplexen Zahlen. Wenn für α, β, $\gamma \in \mathbb{K}$, $\gamma \neq 0$, gilt

$$\begin{array}{ll} (1) & (\alpha + \beta) \in \mathbb{K} \\ (2) & (\alpha - \beta) \in \mathbb{K} \\ (3) & (\alpha \cdot \beta) \in \mathbb{K} \\ (4) & (\frac{\alpha}{\gamma}) \in \mathbb{K}, \end{array}$$

dann nennt man $\mathbb{K}$ einen ***Zahlkörper****.*

Definition 1.3 (algebraische Zahl, Konjugierten)

- *Eine Zahl $\rho \in \mathbb{C}$, die Wurzel eines Polynoms*

$$f(x) = f_0 + f_1 \cdot x + \ldots + f_{d-1} \cdot x^{d-1} + x^d \in \mathbb{Q}[x]$$

  ist, heißt ***algebraische Zahl****.*

- *Ist $f(x)$ irreduzibel über $\mathbb{Q}$, dann heißt d der* ***Grad*** *der algebraischen Zahl $\rho =: \rho^{(1)}$, und die weiteren komplexen Wurzeln $\rho^{(2)}, \rho^{(3)}, \ldots, \rho^{(d)}$ von $f(x)$ heißen die* ***Konjugierten*** *von ρ.*

Definition 1.4 (Polynomdiskriminante) [Po93, Kapitel IV.1, (30)]
Für ein Polynom $f(x)$ vom Grad d mit den Wurzeln $\rho^{(1)}, \ldots, \rho^{(d)}$ heißt die Zahl

$$d(f) = \prod_{1 \leq i < j \leq d} \left(\rho^{(i)} - \rho^{(j)}\right)^2$$

Diskriminante von *f.*

Lemma 1.5 ($\mathbb{Q}(\rho)$)
Sei ρ eine algebraische Zahl vom Grad d. Dann bildet die Menge

$$\mathbb{Q}(\rho) := \{\alpha \mid \alpha = a_0 + a_1 \cdot \rho + \ldots + a_{d-1} \cdot \rho^{d-1},\ a_i \in \mathbb{Q},\ i = 0, \ldots, d-1\} \qquad (1.1)$$

einen Zahlkörper.

Bemerkung 1.6 ($\mathbb{Q}$–Vektorraum)
Ist ρ eine algebraische Zahl vom Grad d, so ist $\mathbb{Q}(\rho)$ ein d-dimensionaler Vektorraum über $\mathbb{Q}$.

Definition 1.7 (ganze algebraische Zahl)
Ist $f(x) = f_0 + f_1 \cdot x + \ldots + f_{d-1} \cdot x^{d-1} + x^d \in \mathbb{Z}[x]$ irreduzibel über $\mathbb{Q}$ und ist ρ eine Wurzel von $f(x)$, dann heißt ρ eine ***ganze algebraische Zahl*** *vom Grad d.*

Lemma 1.8 ($\mathcal{O}_\rho$, $\mathbb{Z}[\rho]$)
Sei ρ eine ganze algebraische Zahl vom Grad d. Dann bildet sowohl die Menge $\mathcal{O}_\rho$ der ganzen algebraischen Zahlen in $\mathbb{Q}(\rho)$, als auch die Menge $\mathbb{Z}[\rho]$,

$$\mathbb{Z}[\rho] := \{\alpha \mid \alpha = a_0 + a_1 \cdot \rho + \ldots + a_{d-1} \cdot \rho^{d-1},\ a_i \in \mathbb{Z},\ i = 0, \ldots, d-1\} \subseteq \mathcal{O}_\rho \subseteq \mathbb{Q}(\rho),$$

einen Integritätsbereich, d.h. einen kommutativen, nullteilerfreien Ring mit 1. $\mathcal{O}_\rho$ nennt man auch ***Ganzheitsring*** *von $\mathbb{Q}(\rho)$.*

Bemerkung 1.9 (Darstellung eines Körperelementes) [Ch93, Kapitel 4.2.2]

Jedes Element α aus $\mathbb{Q}(\rho)$ läßt sich eindeutig als

$$\alpha = \frac{\sum\limits_{i=0}^{d-1} a_i \cdot \rho^i}{z}$$

mit $z > 0$, $a_i \in \mathbb{Z}$ und $\mathrm{ggT}(a_0, \ldots, a_{d-1}, z) = 1$ darstellen.

Definition 1.10 (Ganzheitsbasis)
*Sei ρ eine algebraische Zahl vom Grad d. Eine Menge von d Zahlen $\omega_1, \omega_2, \ldots, \omega_d$, alle $\omega_i \in \mathcal{O}_\rho$, heißt **Ganzheitsbasis** von $\mathbb{Q}(\rho)$, wenn sich jede Zahl $\alpha \in \mathcal{O}_\rho$ eindeutig in der Form $\alpha = \sum\limits_{i=1}^{d} z_i \cdot \omega_i$ mit Koeffizienten $z_i \in \mathbb{Z}$ darstellen läßt.*

Im weiteren sollen folgende Voraussetzungen gelten :

- $\rho = \rho^{(1)}$ ist Nullstelle des über $\mathbb{Q}$ irreduziblen Polynoms $f_0 + f_1 \cdot x + \ldots + f_{d-1} \cdot x^{d-1} + x^d$ und somit eine ganze algebraische Zahl vom Grad d,
- $\rho^{(2)}, \rho^{(3)}, \ldots, \rho^{(d)}$ sind die Konjugierten von ρ.
- $\mathbb{K} := \mathbb{Q}(\rho)$ ist der von ρ erzeugte Zahlkörper.
- $\mathcal{O}_\rho$ ist der Ring der ganzen algebraischen Zahlen in $\mathbb{Q}(\rho)$.

Definition 1.11 (Körperdiskriminante) [Ch93, Prop. 4.4.1 und Def. 4.4.3]
Sei $\{\omega_1, \ldots, \omega_d\}$ eine Basis von $\mathcal{O}_\rho$ und sei $w_{ij} = \omega_j^{(i)}$ die i–te Konjugierte von ω_j. Dann heißt das Quadrat der Determinante der Matrix (w_{ij})

$$d(\mathbb{K}) := det(w_{ij})^2$$

***Diskriminante** des Körpers $\mathbb{K}$.*

Definition 1.12 (Einheit)
*Eine ganze algebraische Zahl $\epsilon \in \mathcal{O}_\rho$ heißt **Einheit**, wenn es ein $\eta \in \mathcal{O}_\rho$ gibt mit*

$$\epsilon \cdot \eta = 1.$$

Satz 1.13 (Konjugierte algebraischer Zahlen)
Sei $\alpha = a_0 + a_1 \cdot \rho + \ldots + a_{d-1} \cdot \rho^{d-1} \in \mathbb{Q}(\rho)$, dann sind

$$\alpha^{(i)} = a_0 + a_1 \cdot \rho^{(i)} + \ldots + a_{d-1} \cdot {\rho^{(i)}}^{d-1} \quad , i = 2, \ldots, d$$

die Konjugierten von α.

Definition 1.14 (Norm)
Für eine algebraische Zahl α vom Grad d heißt die Zahl

$$N(\alpha) := \prod_{i=1}^{d} \alpha^{(i)}$$

die ***Norm*** *von α.*

Satz 1.15 (Normeigenschaften)
Seien α, β algebraische Zahlen, dann gilt

1. *$N(\alpha) \in \mathbb{Q}$.*
2. *Ist $\alpha \in \mathcal{O}_\rho$, dann ist $N(\alpha) \in \mathbb{Z}$.*
3. *Die Norm ist multiplikativ, d.h.*
$$N(\alpha \cdot \beta) = N(\alpha) \cdot N(\beta).$$
4. *Eine ganze algebraische Zahl $\epsilon \in \mathcal{O}_\rho$ ist genau dann eine Einheit, wenn $N(\epsilon) = \pm 1$.*

Lemma 1.16 (Diskriminante eines Polynoms) [Po93, Kapitel IV.1, (31)]
Für die Diskriminante von f gilt

$$d(f) = (-1)^{d(d+1)/2} N(f'(\rho)).$$

1.4.3 Ideale

Definition 1.17 (Ideal)
Eine Teilmenge $\mathcal{I} \subset \mathcal{O}_\rho$ heißt ***Ideal*** *in $\mathcal{O}_\rho$, wenn für alle $\alpha, \beta \in \mathcal{I}$ und alle $\lambda, \mu \in \mathcal{O}_\rho$ gilt*

$$\lambda \cdot \alpha + \mu \cdot \beta \in \mathcal{I}.$$

Bemerkung 1.18 (Schreibweise)
Seien $\alpha_1, \ldots, \alpha_k \in \mathcal{O}_\rho$. Dann nennt man

$$\mathcal{I} = \{\alpha \mid \alpha = \lambda_1 \cdot \alpha_1 + \ldots + \lambda_k \cdot \alpha_k,\ \lambda_i \in \mathcal{O}_\rho\}$$

das von $\alpha_1, \ldots, \alpha_k$ erzeugte Ideal, und man schreibt

$$\mathcal{I} =< \alpha_1,\ \alpha_2, \ldots,\ \alpha_k > .$$

Definition 1.19 (Hauptideal)
Ein Ideal $\mathcal{I}$ heißt ***Hauptideal****, wenn es ein $\alpha \in \mathcal{I}$ gibt mit*

$$\mathcal{I} =< \alpha > .$$

Definition 1.20 (Basis eines Ideals)
Sei $\mathcal{I}$ ein Ideal in $\mathcal{O}_\rho$, $\mathcal{I} \neq< 0 >$. Dann existiert eine Menge B von d Zahlen $\alpha_1, \ldots, \alpha_d \in \mathcal{O}_\rho$, so daß jede Zahl $\alpha = z_1 \cdot \alpha_1 + \ldots + z_d \cdot \alpha_d \in \mathcal{I}$ genau einmal dargestellt wird, wenn die Zahlen z_i, $i = 1, \ldots, d$, alle Zahlen aus $\mathbb{Z}$ durchlaufen. B heißt ***Basis*** *des Ideals.*

Definition 1.21 (Norm eines Ideals)
Sei $\mathcal{I}$ ein Ideal in $\mathcal{O}_\rho$, $\{\omega_1, \omega_2, \ldots, \omega_d\}$ eine Ganzheitsbasis von $\mathbb{K}$ und $\{\alpha_1, \alpha_2, \ldots, \alpha_d\}$ eine Basis von $\mathcal{I}$ mit

$$\alpha_i = \sum_{j=1}^{d} a_{i,j} \cdot \omega_j,\ a_{i,j} \in \mathbb{Z},$$

dann heißt der Absolutbetrag der Determinante der Matrix $(a_{i,j})$, $i, j = 1, \ldots, d$, ***Norm*** *von $\mathcal{I}$:*

$$\mathcal{N}(\mathcal{I}) := |det(a_{i,j})|.$$

Satz 1.22 (Norm von Hauptidealen)

1. *Ist $\mathcal{I} =< \alpha >$ ein Hauptideal in $\mathcal{O}_\rho$, dann gilt*

$$\mathcal{N}(\mathcal{I}) = |N(\alpha)|.$$

2. *Ist $\mathcal{I} =< p >$, $p \in \mathbb{P}$, so gilt*

$$\mathcal{N}(\mathcal{I}) = p^d.$$

Definition 1.23 (Idealprodukt, teilbar, Primideal)
Seien $\mathcal{A} =< \alpha_1, \alpha_2, \ldots, \alpha_k >$ und $\mathcal{B} =< \beta_1, \beta_2, \ldots, \beta_r >$ Ideale in $\mathcal{O}_\rho$.

1. *Unter dem* ***Produkt*** *der Ideale $\mathcal{A}$ und $\mathcal{B}$ versteht man das Ideal*

$$\mathcal{A} \cdot \mathcal{B} :=< \alpha_1 \cdot \beta_1, \ldots, \alpha_1 \cdot \beta_r, \alpha_2 \cdot \beta_1, \ldots, \alpha_k \cdot \beta_r > .$$

2. *$\mathcal{A}$ heißt* ***teilbar*** *durch $\mathcal{B}$, wenn es ein Ideal $\mathcal{C}$ gibt mit*

$$\mathcal{A} = \mathcal{B} \cdot \mathcal{C}.$$

3. *Ein Ideal $\mathcal{P}$ heißt* ***Primideal****, wenn gilt*

   (a) *$\mathcal{P} \neq< 1 >$ und aus*

(b) $\mathcal{A} \mid \mathcal{P}$ *folgt*
$\mathcal{A} =< 1 >$ *oder* $\mathcal{A} = \mathcal{P}$.

Lemma 1.24 (Ideal und Norm)
Sei $\mathcal{I}$ *ein Ideal in* $\mathcal{O}_\rho$.

1. *Wenn die Norm* $\mathcal{N}(\mathcal{I})$ *eine Primzahl ist, so ist* $\mathcal{I}$ *ein Primideal.*
2. $\mathcal{N}(\mathcal{I}) \in \mathcal{I}$.
3. $\mathcal{N}(\mathcal{I}) = \left| {}^{\mathcal{O}_\rho}/_{\mathcal{I}} \right|$.
4. *Wenn* $\mathcal{I}$ *ein Primideal ist, so gilt*
   (a) $p \in \mathcal{I}$ *für genau eine Primzahl* $p \in \mathbb{P}$
   (b) $\mathcal{N}(\mathcal{I}) = p^m$, *für ein* $m \leq d$.

Satz 1.25 (Fundamentalsatz der Idealtheorie)
Jedes von $< 0 >$ *und* $< 1 >$ *verschiedene Ideal in* $\mathcal{O}_\rho$ *läßt sich, bis auf die Reihenfolge der Faktoren, auf eindeutige Weise als Produkt von Primidealen darstellen.*

Definition 1.26 (Index von ρ in $\mathcal{O}_\rho$) [Ch93, Prop. 4.4.4]
Die Zahl $[\,\mathcal{O}_\rho : \mathbb{Z}[\rho]\,]$ *mit*

$$[\,\mathcal{O}_\rho : \mathbb{Z}[\rho]\,]^2 = \frac{d(f)}{d(\mathbb{K})}$$

heißt ***Index*** *von* ρ *in* $\mathcal{O}_\rho$.

Satz 1.27 (Zerlegung von $p\mathcal{O}_\rho$) [Ch93, Theorem 4.8.13]
Sei $p \in \mathbb{P}$ *mit* $p \nmid [\,\mathcal{O}_\rho : \mathbb{Z}[\rho]\,]$. *Zerfällt* $f(x)$ *über* $\mathbb{Z}/p\mathbb{Z}$ *in*

$$f(x) \equiv \prod_{i=1}^{r} f_i(x)^{e_i} \quad (mod\ p),$$

wobei $f_i(x) \in \mathbb{Z}/p\mathbb{Z}[x]$ *irreduzibel über* $\mathbb{Z}/p\mathbb{Z}$ *ist, dann zerfällt das Hauptideal* $p\mathcal{O}_\rho$ *in*

$$p\mathcal{O}_\rho = \prod_{i=1}^{r} \mathcal{P}_i^{e_i},$$

wobei $\mathcal{P}_i$ *das von* $< p, f_i(\rho) >$ *erzeugte Primideal ist.*

1.4.4 Endliche Körper und zyklische Gruppen

Die folgenden Sätze zu endlichen Körpern habe ich der Arbeit [Ho73, §9 bzw. §59] entnommen. Das neutrale Element einer Gruppe wird im weiteren mit e bezeichnet.

Definition 1.28 (Ordnung eines Elementes)
Ist g ein Element der Gruppe G und $g^m = e$ nur für $m = 0$, so heißt $g \in G$ ein Element von unendlicher Ordnung. Ist andernfalls m der kleinste positive Exponent mit

$$g^m = e,$$

so sagen wir, g habe die Ordnung m.

Satz 1.29
Ist G eine endliche Gruppe der Ordnung m und g ein Element aus G, so gilt

$$g^m = e.$$

Satz 1.30 (Untergruppen zyklischer Gruppen)
Es sei G eine von g erzeugte zyklische Gruppe. Dann gelten folgende Aussagen.

- *Jede Untergruppe U von G ist zyklisch.*
- *Hat $G = \{e, g, \ldots, g^{m-1}\}$ die Ordnung m, so gibt es zu jeder natürlichen Zahl d, die m teilt, genau eine Untergruppe U_d der Ordnung d von G; sie ist zyklisch und wird von*

  $$g^{\frac{m}{d}}$$

  erzeugt.

Satz 1.31 ($\mathbb{K}^*$)
Die multiplikative Gruppe $\mathbb{K}^*$ eines endlichen Körpers ist zyklisch.

Satz 1.32 (endliche Untergruppen)
Es sei $\mathbb{K}$ ein Körper und G eine endliche Untergruppe der multiplikativen Gruppe $\mathbb{K}^*$. Dann ist G zyklisch.

Kapitel 2

Die Grundidee des *Number Field Sieve*

In diesem Kapitel möchte ich das Grundgerüst des *Number Field Sieve* Algorithmus beschreiben. Im ersten Abschnitt erläutere ich dazu das Prinzip des Faktorisierens mit quadratischen Kongruenzen. Die beiden folgenden Passagen zeigen, wie diese Kongruenzen im NFS generiert werden. Im Absatz 2.4 gehe ich auf die Berechnung der Wurzeln aus den beiden Quadraten ein. Abgeschlossen wird dieses Kapitel von den Ergebnissen einer heuristischen Laufzeitanalyse für den *Number Field Sieve* Algorithmus.

2.1 Faktorisieren mit quadratischen Kongruenzen

Schon Fermat und Legendre haben zum Faktorisieren natürlicher Zahlen quadratische Kongruenzen benutzt. Zu erklären, was quadratische Kongruenzen sind, und wie man mit ihrer Hilfe faktorisieren kann, ist das Ziel dieses Abschnittes.

Sei n die zu faktorisierende Zahl. Findet man Lösungen $(x, y) \in \mathbb{Z}^2$, $(x, y) \neq (0, 0)$, für eine Kongruenz der Form

$$x^2 \equiv y^2 \pmod{n}, \tag{2.1}$$

so lassen sich daraus Faktoren von n bestimmen, denn aus (2.1) folgt

$$x^2 - y^2 = (x + y) \cdot (x - y) \equiv 0 \pmod{n}. \tag{2.2}$$

Somit teilt n das Produkt $(x + y) \cdot (x - y)$. Gilt nun weiter noch

$$x \not\equiv \pm y \pmod{n}, \tag{2.3}$$

so geht n weder in $x - y$ noch in $x + y$, wohl aber, wie gesehen, in deren Produkt ganz auf. Somit liefert der größte gemeinsame Teiler $\mathrm{ggT}(n, x-y)$ bzw. $\mathrm{ggT}(n, x+y)$ jeweils einen nichttrivialen Teiler von n.

Beispiel 2.1 (quadratische Kongruenz)
Sei $n = 391$, $x = 87$ und $y = 189$. Es gilt

$$x^2 \equiv 87^2 \equiv 140 \equiv 189^2 \equiv y^2 \pmod{391}.$$

Die Bildung der beiden größten gemeinsamen Teiler liefert

1. $\mathrm{ggT}(x - y, n) = \mathrm{ggT}(-102, 391) = 17$ und

2. $\mathrm{ggT}(x + y, n) = \mathrm{ggT}(276, 391) = 23$

und somit die beiden Teiler 17 und 23 von 391. □

Bemerkung 2.2 (quadratische Kongruenz für zusammengesetzte Zahlen)

1. Für ungerade, zusammengesetzte Zahlen, die keine Primzahlpotenzen sind, läßt sich immer eine Kongruenz der Form (2.1) mit der Nebenbedingung (2.3) finden. Denn seien $n_1,\ n_2 \in \mathbb{Z}$ mit $n = n_1 \cdot n_2$ zwei nichttriviale Teiler von n, so erhält man durch Setzen von

   $$x := \frac{n_1 + n_2}{2} \quad \text{und} \quad y := \frac{n_1 - n_2}{2}$$

   wegen

   $$\begin{array}{lll} 1.) & x + y & = n_1 \\ 2.) & x - y & = n_2 \quad \text{und} \\ 3.) & x^2 - y^2 & = n_1 \cdot n_2 = n \end{array}$$

   eine Kongruenz der gewünschten Form.

2. Im weiteren Verlauf dieser Arbeit werde ich davon ausgehen, daß n eine ungerade, zusammengesetzte Zahl, aber keine Primzahlpotenz ist.

   Die Behandlung der ausgeschlossenen Fälle ist sehr einfach. Ist n gerade, so ist natürlich 2 ein Primteiler. Dividiert man die größtmögliche Zweierpotenz aus n heraus, so erhält man eine ungerade Zahl, die mittels quadratischer Kongruenzen faktorisiert werden kann, falls sie weder eins noch eine Primzahl ist.

   Im Falle, daß n eine Primzahlpotenz ist, wird die Bedingung (2.3) aufgrund der Art und Weise, wie im NFS x und y berechnet werden, nie erfüllt sein. Allerdings gibt es einfache Algorithmen, die überprüfen, ob eine Zahl eine Primzahlpotenz ist, und gegebenenfalls die Wurzel ziehen. Ein solches Verfahren werde ich im Anhang A.2 auf Seite 161 beschreiben. □

Eine Vielzahl von Faktorisierungsalgorithmen, wie z.B. Dixon's Random Square Methode ([Br88, Kapitel 8.1]), CFRAC ([Br88, Kapitel 11]) oder das Quadratische Sieb ([Br88, Kapitel 8]), bedienen sich solcher quadratischen Kongruenzen. Der Unterschied zwischen den einzelnen Verfahren besteht darin, wie diese Kongruenzen erzeugt werden. Auch das NFS ist ein Algorithmus, der die Faktoren durch solche Kongruenzen bestimmt. Auf welche Weise die Kongruenzengenerierung im NFS erfolgt, werde ich im nächsten Abschnitt vorstellen.

2.2 Konstruktion quadratischer Kongruenzen

Ziel dieses Abschnittes ist es aufzuzeigen, wie im NFS die zum Berechnen der Faktoren benötigte quadratische Kongruenz generiert wird. Für den theoretischen Hintergrund des Algorithmus verweise ich auf [BuLePo93].

Sei $f(x) = f_D \cdot x^D + \ldots + f_0 \in \mathbb{Z}[x]$ ein über $\mathbb{Q}$ irreduzibles Polynom vom Grade $D > 1$ und $m \in \mathbb{N}$, so daß

$$f(m) \equiv 0 \pmod{n}, \text{ aber } f(m) \neq 0. \tag{2.4}$$

Sei weiterhin $\rho \in \mathbb{C}$ eine komplexe Wurzel von $f(x)$, so bildet $\mathbb{Q}[x]/f(x) \cong \mathbb{Q}(\rho)$ einen Zahlkörper.

Definiert man das Polynom $g(x)$ durch

$$g(x) := f\left(\frac{x}{f_D}\right) \cdot {f_D}^{D-1}, \tag{2.5}$$

so erhält man folgende Eigenschaften (siehe [BuLePo93, Kapitel 12] oder [BeLe93, Kapitel 3]):

Lemma 2.3

1. *$g(x) \in \mathbb{Z}[x]$ ist normiert und irreduzibel.*
2. *$\omega = f_D \cdot \rho$ ist eine Wurzel von $g(x)$.*
3. *Die beiden Zahlkörper $\mathbb{Q}(\rho)$ und $\mathbb{Q}(\omega)$ sind gleich.*
4. *Die Abbildung $\psi\colon \mathbb{Z}[\omega] \to \mathbb{Z}/n\mathbb{Z}$, die ω auf $f_D \cdot m \pmod{n}$ abbildet, ist ein Ringhomomorphismus.* □

Gelingt es eine Menge S von Paaren $(a, b) \in \mathbb{Z}^2$ derart zu bestimmen, daß

$$|S| \equiv 0 \pmod{2}, \tag{2.6}$$

also $|S|$ gerade ist, und

$$\prod_{(a,b)\in S} (a + bm) = \tilde{x}^2 \in \mathbb{Z} \tag{2.7}$$

$$\prod_{(a,b)\in S} (a + b\rho) = \tilde{\delta}^2,\ \tilde{\delta} \in \mathbb{Q}(\rho) \tag{2.8}$$

gilt, dann ist

$$\begin{aligned} {f_D}^{|S|} \cdot \prod_{(a,b)\in S} (a + b\rho) &= \prod_{(a,b)\in S} (f_D \cdot a + b\omega) \\ &:= \delta^2 \in \mathbb{Z}[\omega] \end{aligned} \tag{2.9}$$

ein Quadrat in $\mathbb{Z}[\omega]$. Liegt δ selbst in $\mathbb{Z}[\omega]$, so ergibt sich unter Ausnutzung der Homomorphieeigenschaften folgende quadratische Kongruenz

$$\begin{aligned}
y^2 := \psi(\delta)^2 &\equiv \psi(\delta^2) \pmod n \\
&\equiv \psi\left(\prod_{(a,b)\in S} (f_D \cdot a + b\omega)\right) \pmod n \\
&\equiv \prod_{(a,b)\in S} \psi\left(f_D \cdot a + b\omega\right) \pmod n \\
&\equiv \prod_{(a,b)\in S} (f_D \cdot a + b \cdot f_D \cdot m) \pmod n \\
&\equiv f_D^{|S|} \cdot \prod_{(a,b)\in S} (a + bm) \pmod n \\
&= f_D^{|S|} \cdot \tilde{x}^2 \\
&=: x^2.
\end{aligned}$$

Somit muß man nur noch die Wurzeln aus x^2 in $\mathbb{Z}$ und aus δ^2 in $\mathbb{Z}[\omega]$ ziehen, um eine Lösung für die Kongruenz (2.1) zu erhalten.
Der Grundalgorithmus zum Faktorisieren mit dem NFS sieht damit wie folgt aus:

Algorithmus 2.4 (NFS Basisalgorithmus)

NFS

EINGABE: $n \in \mathbb{N}$ zusammengesetzt, ungerade, keine Primzahlpotenz
AUSGABE: $n_1,\ n_2 \in \mathbb{N}_{>1}$ mit $n_1 \cdot n_2 = n$

(1) Berechne ein über $\mathbb{Q}$ irreduzibles Polynom $f(x) \in \mathbb{Z}[x]$ und eine ganze Zahl m mit der Eigenschaft (2.4).
(2) Berechne die Menge S von Paaren $(a, b) \in \mathbb{Z}$ mit den Eigenschaften (2.6) bis (2.8).
(3) Berechne die Wurzel x aus ${f_D}^{|S|} \cdot \prod_{(a,b)\in S} (a + bm)$ in $\mathbb{Z}$
sowie die Wurzel δ aus $\prod_{(a,b)\in S} (f_D \cdot a + b\omega)$ in $\mathbb{Z}[\omega]$ und
setze $y := \psi(\delta)$.
(4) Berechne $n_1 := \mathrm{ggT}(x - y, n)$ und $n_2 := \mathrm{ggT}(x + y, n)$.
(5) Ist n_1 und somit auch n_2 ein trivialer Teiler von n, so wiederhole den Algorithmus ab Schritt (2) mit der Berechnung einer neuen Menge S.

Bemerkung 2.5 (Erfolgswahrscheinlichkeit)
Der NFS Basisalgorithmus konstruiert zwar Kongruenzen der Form (2.1), die Bedingung (2.3) muß dabei aber nicht erfüllt sein. Zur weiteren Betrachtung gehe ich

davon aus, daß sich die durch den NFS Algorithmus generierten Lösungen (x, y) wie zufällig erzeugte Lösungen verhalten. Dies konnte bisher noch nicht bewiesen werden, aber die Praxis zeigt, daß diese Annahme wohl korrekt ist (siehe Kapitel 5.5).
Die Wahrscheinlichkeit, einen nichttrivialen Teiler von n gefunden zu haben, ist dann größer gleich $1/2$. Zum Beweis betrachten wir zuerst einmal die Situation, daß n das Produkt zweier Primfaktoren n_1 und n_2 ist. Dann gibt es für diese beiden folgende Möglichkeiten, sich auf $(x - y)$ bzw. $(x + y)$ zu verteilen.

Tabelle 2.1: Verteilung der Primfaktoren

$x + y$	$x - y$
$-$	n_1, n_2
n_1	n_2
n_2	n_1
n_1, n_2	$-$

Die Fälle 2 und 3 liefern im Gegensatz zu den Fällen 1 und 4 nichttriviale Teiler. Unter der Voraussetzung, daß sich (x, y) wie Zufallslösungen verhalten, kommen alle Fälle gleich oft vor. Somit ist die Wahrscheinlichkeit, einen nichttrivialen Teiler berechnet zu haben, gleich $1/2$.
Ist n aus drei verschiedenen Primfaktoren zusammengesetzt, so kann man durch analoge Argumentation leicht zeigen, daß in diesem Falle die Wahrscheinlichkeit, einen nichttrivialen Teiler gefunden zu haben, $3/4$ beträgt. Besitzt n noch mehr Primfaktoren, so erhöht sich die Wahrscheinlichkeit für nichttriviale Faktoren weiter. Insgesamt ist die Erfolgswahrscheinlichkeit also größer gleich $1/2$.
Somit wird mit hoher Wahrscheinlichkeit schon nach wenigen Durchläufen der Schleife (2) – (5) des NFS Basisalgorithmus ein Faktor gefunden. □

Bemerkung 2.6 (ggT–Bedingung)
Bei der Berechnung der Menge S kann man sich auf solche Paare (a, b) zurückziehen, für die $\mathrm{ggT}(a, b) = 1$ ist, denn sei $\tilde{a} = a/\mathrm{ggT}(a, b)$ und $\tilde{b} = b/\mathrm{ggT}(a, b)$, so gilt

$$\begin{aligned}
\mathrm{ggT}(a, b) \cdot f_D \cdot (\tilde{a} + \tilde{b} \cdot m) &= f_D \cdot (a + bm) && (2.10)\\
&\equiv \psi(f_D \cdot a + b\omega) && (2.11)\\
&\equiv \psi\left(\mathrm{ggT}(a, b) \cdot (f_D \cdot \tilde{a} + \tilde{b}\omega)\right) && (2.12)\\
&\equiv \mathrm{ggT}(a, b) \cdot \psi(f_D \cdot \tilde{a} + \tilde{b}\omega) \pmod{n} && (2.13)
\end{aligned}$$

Dividiert man beide Seiten der Kongruenz durch $\mathrm{ggT}(a, b)$ modulo n, was natürlich nur möglich ist, wenn $\mathrm{ggT}(a, b)$ und n teilerfremd sind, so erhält man die gleiche Kongruenz, die schon das Paar $(\tilde{a}, \tilde{b})$ erzeugt. Somit liefern Paare mit ggT größer 1 keinen positiven Beitrag in Hinblick auf die Bedingung (2.3). □

2.3 Konstruktion der Quadrate

In diesem Abschnitt werde ich beschreiben, wie die Menge S aus Schritt (2) des NFS Basisalgorithmus (Algorithmus 2.4) berechnet werden kann.
Die Generierung dieser Menge läuft im wesentlichen in zwei Schritten ab. Zuerst werden viele Paare $(a, b) \in \mathbb{Z}^2$, die in gewissem Sinne *günstig* zum Bilden der gesuchten Quadrate sind, bestimmt. Im zweiten Schritt wird aus diesen Paaren die Teilmenge S gebildet, die gerade die Eigenschaften (2.6) bis (2.8) besitzt.

2.3.1 Generierung der Relationen

Zur Konstruktion einer Menge T günstiger Paare (a, b) werden, analog zu anderen Faktorisierungsalgorithmen, wie z.B. dem Quadratischen Sieb, Faktorbasen verwendet. Dahinter verbirgt sich folgende Idee: Will man aus einer Menge von Zahlen solche auswählen, die miteinander multipliziert ein Quadrat ergeben, so ist es von Vorteil, wenn man die Primfaktorzerlegung dieser Zahlen kennt. Ist die Menge der in Frage kommenden Zahlen groß genug, so kann man sich zur Berechnung der Primfaktorzerlegungen auf eine relativ kleine Teilmenge der Primzahlen, eine sogenannte Faktorbasis, zurückziehen und nur Zahlen betrachten, die über der Faktorbasis zerfallen. Findet man genügend viele solcher Zahlen, so läßt sich daraus leicht eine Teilmenge auswählen, die zu einem Quadrat führt.
Da im NFS sowohl ein rationales als auch ein algebraisches Quadrat generiert werden müssen, benutzt man zwei Faktorbasen. Die erste, die ich im folgenden *rationale* Faktorbasis nennen werde, besteht aus den kleinsten $B1$ Primzahlen:

$$FB_R := \{p_1, \ldots, p_{B1}\}. \tag{2.14}$$

Die zweite Faktorbasis, im weiteren *algebraische* Faktorbasis genannt, besteht aus den ersten $B2$ Paaren

$$FB_A := \{(p_1, cp_1), \ldots (p_{B2}, cp_{B2}) \mid (p_i, cp_i) \in R,\ i = 1, \ldots, B2\}, \tag{2.15}$$

wobei

$$\begin{aligned} R = \quad & \{(p, cp) \mid f(cp) \equiv 0 \bmod p,\ p \in \mathbb{P},\ p \not| f_D,\ cp \in \{0, \ldots, p-1\}\} \\ \cup \quad & \{(p, \infty) \mid p \in \mathbb{P},\ p \mid f_D\}. \end{aligned} \tag{2.16}$$

Die erste Komponente p eines Paares (p, cp) heißt auch *Norm* des Faktorbasiselementes.
Setzt man

$$N(a, b) := f\left(\frac{-a}{b}\right) \cdot (-b)^D, \tag{2.17}$$

so kann man definieren

Definition 2.7 (gute Paare, Relation)
Ein Paar $(a,b) \in \mathbb{Z}^2$ *heißt* gutes Paar *oder* ***Relation****, wenn für gegebene Faktorbasen* FB_R *und* FB_A *folgende drei Bedingungen erfüllt sind:*

$$\begin{aligned} \mathrm{ggT}(a,b) &= 1 && (2.18)\\ a+bm &= \prod_{p\in FB_R} p^{e_p},\ e_p \in \mathbb{N}_0 && (2.19)\\ |N(a,b)| &= \prod_{(p,cp)\in FB_A} p^{f_{(p,cp)}},\ f_{(p,cp)} \in \mathbb{N}_0. && (2.20) \end{aligned}$$

□

Bemerkung 2.8 (Schreibweise)
Für Paare (a,b), die die Bedingung (2.19) oder (2.20) bzw. beide erfüllen, schreibe ich im weiteren: das Paar (a,b) zerfällt über der rationalen Faktorbasis, der algebraischen Faktorbasis bzw. über den Faktorbasen. □

Zur Berechnung der Menge T, bestehend aus genügend vielen guten Paaren, gibt man sich eine Teilmenge AB von $\mathbb{Z}^2$ mit

$$AB = \{(a,b) \in \mathbb{Z}^2 \mid A_{min} \leq a \leq A_{max} \text{ und } 1 \leq b \leq B_{max}\}$$

vor und untersucht für alle $(a,b) \in AB$, ob die drei Bedingungen (2.18) bis (2.20) erfüllt sind. Der einfachste Weg ist das sukzessive Überprüfen der Bedingungen für jedes Paar: Berechne zuerst für ein festes Paar (a,b) den $\mathrm{ggT}(a,b)$. Ist dieser 1, so zerlege $a+bm$ über der rationalen Faktorbasis FB_R und, falls dies möglich ist, so zerlege auch $N(a,b)$ über der algebraischen Faktorbasis FB_A. Gelingt auch dies, so kann man (a,b) zu T hinzufügen. Diese Art der Berechnung von T ist allerdings sehr laufzeitintensiv, da im ersten Schritt sehr viele teure ggT-Berechnungen und in den beiden Zerlegungsschritten viele Divisionen durchgeführt werden müssen.
Im folgenden werde ich effizientere Methoden vorstellen, die ohne explizite ggT--Berechnungen auskommen und die teuren Divisionen zum großen Teil durch billige Additionen ersetzen. Dabei werde ich davon ausgehen, daß das Rechteck AB der zu untersuchenden Paare zeilenweise bearbeitet wird, d.h., wir halten ein $b \in \{1,\ldots,B_{max}\}$ fest und suchen die $a \in \{A_{min},\ldots,A_{max}\}$, für die die Bedingungen (2.18) bis (2.20) erfüllt sind.
Für ein festes b starten wir dazu mit der Menge $T_b := \{(a,b) \mid A_{min} \leq a \leq A_{max}\}$ und eliminieren nacheinander die Paare, die eine der drei Bedingungen verletzen. Im Algorithmus werde ich dazu ein Bitfeld H_b verwenden, wobei $H_b[a] = 0$ bedeutet, daß mindestens eine der drei oben genannten Bedingungen nicht erfüllt ist, das Paar (a,b) also aus T_b entfernt werden muß. Im ersten Schritt streichen wir die Paare (a,b), die die ggT-Bedingung verletzen. Dabei nutzen wir die Tatsache aus, daß genau die a zu b nicht teilerfremd sind, die von einem der Primteiler von b geteilt werden. Seien $q_1,\ldots,q_k$ die verschiedenen Primzahlen, die b teilen, so hat a genau dann einen gemeinsamen Teiler mit b, wenn a ein Vielfaches von mindestens einer dieser Primzahlen q_i, $i = 1,\ldots,k$, ist. Dies machen wir uns folgendermaßen zunutze: Da

wir nur solche a betrachten, die in den Grenzen A_{min} und A_{max} liegen, berechnen wir nacheinander für jeden Primteiler q_i von b den minimalen ganzzahligen Wert a_i im Intervall $[A_{min}, A_{max}]$, der durch q_i teilbar ist. Dies erreicht man durch Berechnung des Abstandes r_i von A_{min} zu a_i. Dieser ergibt sich durch

$$r_i = -A_{min} \ Mod\, q_i, \tag{2.21}$$

wobei mit $Mod\, q_i$ der kleinste nichtnegative Restklassenvertreter bezeichnet wird, also ein Element aus $\{0, \ldots, q_i - 1\}$. a_i läßt sich dann durch

$$a_i = A_{min} + r_i \tag{2.22}$$

berechnen. Die weiteren Werte a, die durch q_i teilbar sind, ergeben sich dann durch sukzessives Addieren von q_i, also

$$a = a_i + l \cdot q_i, \text{ für } l \in \mathbb{N}.$$

Diese Stellen lassen sich also durch *Sieben* mit q_i ermitteln. Den Gesamtalgorithmus zur Eliminierung der bezüglich (2.18) unbrauchbaren Paare (a, b) bezeichne ich mit ggT-*Sieb*.

Algorithmus 2.9 (**ggT-Sieb**)

GGTSIEB

EINGABE: b, A_{min}, A_{max}
AUSGABE: Bitarray H_b mit

$$H_b[a] = \begin{cases} 1 & \text{wenn ggT}(a, b) = 1 \\ 0 & \text{sonst} \end{cases}$$

PHASE 1: VORBERECHNUNG UND INITIALISIERUNG

(1)	Berechne Primfaktoren $q_1, \ldots, q_k$ von b	/* Probedivision */
(2)	**for** $(a := A_{min}; a \leq A_{max}; a{+}{+})$ **do**	
(3)	$\quad H_b[a] := 1$	/* Initialisierung des Bitfeldes */
(4)	**od**	

PHASE 2: SIEBPHASE

(5)	**for** $(i := 1; i \leq k; i{+}{+})$ **do**	/* für alle Primteiler von b */
(6)	$\quad r_i := -A_{min} \ Mod\, q_i$	/* Abstand nach (2.21) */
(7)	$\quad a_i := A_{min} + r_i$	/* erste Stelle an der q_i teilt */

Fortsetzung auf Seite 23

```
(8)      for (a := a_i; a ≤ A_max; a := a+q_i)  /* Sieben mit q_i */
           do
(9)        H_b[a] := 0                          /* Streichen der Vielfachen */
(10)     od
(11)   od
```

Beispiel 2.10 (ggT-Sieb)
Dieses Beispiel soll die Arbeitsweise des ggT-Siebes verdeutlichen. Sei $n = 391$ zu faktorisieren. Das den Körper definierende Polynom sei

$$f(x) = x^3 + x^2 - 1,$$

bei $m = 7$, dann gilt $f(7) = 391$. Betrachten wir einmal die Zeile $b = 3$ des Rechtecks AB, definiert durch $A_{min} = -A_{max} = -5$ und $B_{max} = 6$. Da b in diesem Falle eine Primzahl ist, müssen wir nur die Vielfachen von $q_1 = 3$ ausstreichen. Der Abstand r_1 von A_{min} zu der ersten Stelle a_1, an der 3 teilt, ergibt sich zu

$$r_1 = -A_{min}\, Mod\, q_1 = -(-5)\, Mod\, 3 = 2$$

und somit a_1 zu

$$a_1 = A_{min} + r_1 = -5 + 2 = -3.$$

Die weiteren Stellen, an denen 3 teilt, erhält man durch wiederholtes Addieren der 3. Diese Stellen sind hier noch $a = 0$ und $a = 3$. Das Bitarray H_3 hat also nach Aufruf des ggT-Sieb-Algorithmus folgendes Aussehen :

a	-5	-4	-3	-2	-1	0	1	2	3	4	5
H_3	1	1	0	1	1	0	1	1	0	1	1

□

Bemerkung 2.11 (Die Teiler von b)
Die Berechnung der Primfaktoren von b bedingt die Division von b durch einige Primzahlen. In der Praxis ist dies allerdings kein allzu großer Aufwand, denn geht man davon aus, daß B_{max} dort maximal Werte bis 10^8 annimmt, so muß man nur Divisionen mit Primzahlen bis höchstens $\sqrt{10^8} = 10^4$ durchführen. Dies sind etwa 1230 Stück. □

Schauen wir uns einmal an einem Beispiel an, was durch das ggT-Sieb eingespart wird.

Beispiel 2.12 (Operationen des ggT-**Siebes)**
Sei $b = 210 = 2 \cdot 3 \cdot 5 \cdot 7$, $A_{min} = -A_{max} = -1.5 \cdot 10^5$. Der oben beschriebene einfache Algorithmus benötigt somit $3 \cdot 10^5$ ggT-Berechnungen. Das ggT-Sieb erfordert dagegen für die Probedivision 4 Divisionen sowie für die Primzahlen:

2: eine Division und eine Addition zur Bestimmung der ersten Stelle sowie etwa $1.5 \cdot 10^5$ Additionen während des Siebvorganges

3: eine Division und eine Addition zur Bestimmung der ersten Stelle sowie etwa 10^5 Additionen während des Siebvorganges

5: eine Division und eine Addition zur Bestimmung der ersten Stelle sowie etwa $0.6 \cdot 10^5$ Additionen während des Siebvorganges

7: eine Division und eine Addition zur Bestimmung der ersten Stelle sowie etwa $0.4286 \cdot 10^5$ Additionen während des Siebvorganges

Zählt man die Operationen zusammen, so stehen den $3 \cdot 10^5$ teuren ggT-Berechnugen nur 8 Divisionen sowie $3.5286 \cdot 10^5$ Additionen entgegen. Auf einer Sparc-ELC, einer 21 MIPS–Maschine, entspricht dies 1.48 Sekunden gegenüber 0.03 Sekunden, also eine Beschleunigung etwa um den Faktor 49. □

Die Überprüfung der zweiten Bedingung (2.19), ob $a + bm$ über der rationalen Faktorbasis $FB_R = \{p_1, \ldots, p_{B1}\}$ zerfällt, schließt sich als nächstes zu lösendes Problem dem ggT-Sieb an. Hier ist es nun die Aufgabe, schnell die Paare (a, b), für die $a + bm$ nicht über FB_R zerfällt, aus T_b herauszustreichen. Die Trivialidee, für jedes Paar (a, b) Division mit den Elementen der Faktorbasis durchzuführen, ist wegen der Vielzahl von Divisionen ineffizient, denn für jedes (a, b), das (2.18) erfüllt, muß für alle Faktorbasiselemente überprüft werden, ob $a + bm$ geteilt wird.
Das Verfahren, das ich nun vorstellen werde, geht nacheinander von allen Elementen p_i der Faktorbasis aus, sucht sukzessive die Paare (a, b), für die $a + bm$ ein Vielfaches von p_i ist, und bestimmt die größte Potenz von p_i, die in $a + bm$ aufgeht. Dazu wird ein Siebarray S_b mit $S_b[a] = a + bm$ für alle $a \in \{A_{min}, \ldots, A_{max}\}$ initialisiert. Für jedes rationale Faktorbasiselement p_i werden nun die Stellen $A_{min} \le a \le A_{max}$ lokalisiert, wo p_i $(a + bm)$ teilt, also $S_b[a]$ durch p_i geteilt wird. An diesen Stellen wird $S_b[a]$ solange durch p_i dividiert, wie die Division aufgeht. Wird dies für alle Faktorbasiselemente durchgeführt, so ist $a + bm$ genau dann über FB_R zerlegbar, wenn im Siebarray an der Stelle a nur noch eine 1 steht. Zur Bestimmung der Stellen im Siebarray S_b, an denen ein Faktorbasiselement p_i teilt, nutzt man die Tatsache aus, daß, wenn $(a + bm)$ von p_i geteilt wird, auch gilt

$$p_i | (a + l \cdot p_i) + bm, \quad \forall_{l \in \mathbb{Z}} \ . \tag{2.23}$$

Somit kann man ein zum ggT-Sieb ähnliches Verfahren angeben, das die Paare (a, b), die die Bedingung (2.19) nicht erfüllen, errechnet. Ist a_i der kleinste Index größer oder gleich A_{min} mit $p_i | S_b[a_i]$, so findet man die nächsten Stellen a, an denen p_i teilt, wegen (2.23) durch wiederholtes Addieren von p_i, also

$$a = a_i + l \cdot p_i, \text{ für } l \in \mathbb{N}_0.$$

Die Berechnung des Startindex a_i erfolgt analog zur Berechnung des Startindex im ggT-Sieb (siehe (2.21) und (2.22) auf Seite 22). Zuerst wird der Abstand r_i von A_{min}

zu a_i ermittelt. Dieser beträgt

$$r_i = -(A_{min} + bm) \; Mod \, p_i. \tag{2.24}$$

Dann ergibt sich a_i zu

$$a_i = A_{min} + r_i. \tag{2.25}$$

Der Test, welche Paare (a, b) für festes b die Bedingung (2.19) erfüllen, läßt sich also auch mit Hilfe von Siebtechniken realisieren.

Algorithmus 2.13 (rationales Sieb)

RATSIEB

EINGABE: b, A_{min}, A_{max}, H_b, FB_R
AUSGABE: Bitarray $\tilde{H}_b$ mit

$$\tilde{H}_b[a] = \begin{cases} 1, & \text{wenn } H_b[a] = 1 \text{ und } a + bm = \prod_{p \in FB_R} p^{e_p} \\ 0, & \text{sonst} \end{cases}$$

PHASE 1: INITIALISIERUNG

```
for (a := A_min; a ≤ A_max; a ++) do
   H̃_b[a] := H_b[a]                     /* Bitarray */
   if (H_b[a] = 1) then                 /* (2.18) erfüllt? */
      S_b[a] := a + bm                  /* Siebarray */
od
```

PHASE 2: SIEBPHASE

```
foreach (p ∈ FB_R) do                   /* für alle Faktorbasiselemente */
   r_p := −(A_min + b · m) Mod p        /* Abstand nach (2.24) */
   a_p := A_min + r_p                   /* erste Stelle an der p teilt */
   for (a := a_p; a ≤ A_max; a := a+p)
      do
     if (H_b[a] = 1) then               /* Bedingung (2.18) erfüllt? */
        while (p | S_b[a]) do
           S_b[a] := S_b[a]/p           /* Herausdividieren der maxi-
                                           malen Potenz von p */
        od
     fi
   od
od
```

Fortsetzung auf Seite 26

PHASE 3: AUSWERTUNG DER SIEBPHASE

(17) **for** $(a := A_{min}; a \leq A_{max}; a++)$ **do**
(18) **if** $(S_b[a] \neq 1)$ **then** /* (2.19) nicht erfüllt */
(19) $\tilde{H}_b[a] := 0$ /* Ausstreichen des Paares */
(20) **od**
(21) **return** $(\tilde{H}_b)$

Bemerkung 2.14 (rationales Sieb)

- Da nur solche Paare (a, b) von Interesse sind, die alle drei Bedingungen (2.18) bis (2.20) erfüllen, können wir uns bei der Überprüfung von (2.19) auf solche Paare zurückziehen, die auch Forderung (2.18) erfüllen, für die also zu Beginn des rationalen Siebes $H_b[a] = 1$ ist. Dem wird durch die Schritte (2), (3) und (10) des rationalen Siebes Rechnung getragen.
- Mußten bei Verwendung des trivialen Divisionsalgorithmus noch für alle (2.18) erfüllenden Paare Divisionen mit jedem Faktorbasiselement durchgeführt werden, so benötigt das rationale Sieb zur Berechnung des Startindex nur noch eine Division pro Faktorbasiselement und darüber hinaus an Stellen, wo dieses Element teilt, weitere Divisionen zum Herausdividieren der maximalen Potenz. □

Beispiel 2.15 (Fortsetzung von Beispiel 2.10)
Seien $n = 391$, $m = 7$, $FB_R = \{2, 3, 5, 7, 11, 13\}$, $A_{min} = -A_{max} = -5$, $b = 3$. Die folgende Tabelle zeigt, wie sich die Einträge im Siebintervall S_3 während des RATSIEB–Algorithmus ändern. Mit x werden dabei die Stellen gekennzeichnet, an denen das aktuelle Faktorbasiselement zwar teilen würde, die aber im GGTSIEB schon ausgestrichen wurden.

a	-5	-4	-3	-2	-1	0	1	2	3	4	5
H_3	1	1	0	1	1	0	1	1	0	1	1
$S_3[a]$	16	17	-	19	20	-	22	23	-	25	26
$p = 2$	1		x		5		11		x		13
$p = 3$			x			x			x		
$p = 5$					1					1	
$p = 7$						x					
$p = 11$							1				
$p = 13$											1
$\Rightarrow \tilde{H}_3$	1	0	0	0	1	0	1	0	0	1	1

□

Die beiden ersten Bedingungen, die nach Definition 2.7 an ein gutes Paar (a, b) gestellt werden, haben wir jetzt untersucht. Bleibt noch die Eigenschaft zu überprüfen, ob $N(a, b)$ über der algebraischen Faktorbasis zerfällt. Bevor ich nun mit der Beschreibung des Verfahrens fortfahre, möchte ich zuerst erläutern, wieso die Zerlegung von $N(a, b)$ wichtig ist, und was es mit den Elementen der algebraischen Faktorbasis (2.15) auf sich hat.
Für den theoretischen Hintergrund der nachfolgenden Betrachtungen verweise ich auf die Kapitel 5, 7 und 12.6 aus [BuLePo93].
Sei

$$A = \mathbb{Z}[\rho] \cap \mathbb{Z}[\rho^{-1}], \tag{2.26}$$

dann ist A eine Ordnung von $\mathbb{Q}(\rho)$, und die Paare $(p, cp) \in R$, R wie in (2.16) korrespondieren bijektiv zu den Primidealen primer Norm von A.
Unter diesen Voraussetzungen gilt das Lemma

Lemma 2.16 (l_π) ([BuLePo93, Prop. 7.1])
Für jedes Primideal π von A gibt es einen Gruppenhomomorphismus

$$l_\pi : \mathbb{Q}(\rho)^* \to \mathbb{Z}$$

mit

1. $l_\pi(x) \geq 0$ *für alle* $x \neq 0 \in A$;
2. *für* $x \neq 0 \in A$ *gilt:* $l_\pi(x) > 0 \Leftrightarrow x \in \pi$;
3. *für jedes* $x \in \mathbb{Q}(\rho)^*$ *gilt:* $l_\pi(x) = 0$ *für alle bis auf endlich viele* π, *und es gilt*

$$\prod_\pi \mathcal{N}(\pi)^{l_\pi(x)} = |N(x)|.$$

□

Ist π ein Primideal nicht primer Norm, so gilt für $(a, b) \in \mathbb{Z}^2$ mit $\mathrm{ggT}(a, b) = 1$

$$l_\pi(a + b\rho) = 0.$$

Setzt man für Primideale π primer Norm bzw. für die ihnen entsprechenden Paare $(p, cp) \in R$

$$f_{(p,cp)}(a + b\rho) = \begin{cases} l_\pi(a + b\rho), & \text{wenn } cp \neq \infty \\ l_\pi(a + b\rho) + ord_p(f_D), & \text{wenn } cp = \infty, \end{cases} \tag{2.27}$$

so gilt

Lemma 2.17 ($f_{(p,cp)}$) ([BuLePo93, Prop. 5.3])
Ist S eine Menge teilerfremder Paare $(a,b) \in \mathbb{Z}^2$ mit der Eigenschaft, daß

$$\prod_{(a,b)\in S} (a+b\rho) = \eta^2 \in \mathbb{Q}(\rho),\ \eta \in \mathbb{Q}(\rho)$$

ein Quadrat eines Elementes aus $\mathbb{Q}(\rho)$ ist, so gilt für jedes Paar $(p,cp) \in R$

$$\sum_{(a,b)\in S} f_{(p,cp)}(a+b\rho) \equiv 0 \quad (mod\ 2).$$

□

Unter der Verwendung von (2.17) gilt für die $f_{(p,cp)}(a+b\rho)$

$$f_{(p,cp)}(a+b\rho) = \begin{cases} ord_p(N(a,b)), & \text{wenn } a+b\cdot cp \equiv 0 \bmod p,\ cp \neq \infty \\ ord_p(N(a,b)), & \text{wenn } p|b,\ cp = \infty \\ 0, & \text{sonst.} \end{cases} \tag{2.28}$$

Da die Paare (p,cp) aus der algebraischen Faktorbasis jeweils genau einem Primideal entsprechen, sage ich im weiteren auch, die algebraische Faktorbasis besteht aus Primidealen primer Norm. Diese werde ich mit $\pi_{p,cp}$ oder einer Numerierung entsprechend mit π_i bezeichnen.
Die Überprüfung der Bedingung (2.20) kann nun weitestgehend analog zum rationalen Sieb vorgenommen werden. Für ein Paar (p,cp), $cp \neq \infty$, aus der algebraischen Faktorbasis wird die erste Stelle a_i mit

$$p\,|\,(a_i + b\cdot cp) \tag{2.29}$$

lokalisiert. Die Berechnung des Startindex a_i geht daher analog zur Startindexberechnung des rationalen Siebes. Zuerst wird der Abstand r_i von A_{min} zu a_i ermittelt. Dieser beträgt

$$r_i = -(A_{min} + b\cdot cp) \quad Mod\, p. \tag{2.30}$$

Dann ergibt sich a_i zu

$$a_i = A_{min} + r_i. \tag{2.31}$$

Ausgehend von dieser Stelle lassen sich die weiteren aufgrund der Bedingung (2.29) und der daraus resultierenden Folgerung

$$p\,|\,((a+l\cdot p)+b\cdot cp)\,, \quad \forall_{l\in\mathbb{Z}}\,, \tag{2.32}$$

durch wiederholtes Addieren von p erreichen. Die Überprüfung der Bedingung (2.20) läßt sich also auch durch Sieben realisieren.
Ist $cp = \infty$, so muß zuerst überprüft werden, ob $p|b$. Ist dies nicht der Fall, so liefert (p,∞) für dieses b keinen Beitrag. Teilt p jedoch b, so muß für alle $A_{min} \le a \le A_{max}$ die größtmögliche Potenz von p herausdividiert werden.
Diese Betrachtungen führen zu dem folgenden Algorithmus.

Algorithmus 2.18 (algebraisches Sieb)

ALGSIEB

EINGABE: b, A_{min}, A_{max}, H_b, FB_A
AUSGABE: Bitarray $\tilde{H}_b$ mit

$$\tilde{H}_b[a] = \begin{cases} 1, & \text{wenn } H_b[a]=1 \text{ und } |N(a,b)| = \prod_{(p,cp)\in FB_A} p^{f(p,cp)} \\ 0, & \text{sonst} \end{cases}$$

PHASE 1: INITIALISIERUNG DES SIEBARRAYS

(1) **for** $(a := A_{min}; a \leq A_{max}; a\,{+}{+})$ **do**
(2) $\quad\tilde{H}_b[a] := H_b[a]$ /∗ Bitarray ∗/
(3) $\quad$**if** $(H_b[a]=1)$ **then** /∗ (2.18) und (2.19) erfüllt? ∗/
(4) $\quad\quad S_b[a] := |N(a,b)|$ /∗ Siebarray ∗/
(5) **od**

PHASE 2: SIEBPHASE

(6) **foreach** $((p,cp) \in FB_A)$ **do** /∗ für alle Faktorbasiselemente $(p,cp) = (p_i, cp_i)$ ∗/
(7) $\quad$**if** $(cp \neq \infty)$ **then** /∗ normales FB–element ∗/
(8) $\quad\quad r_i := -(A_{min} + b \cdot cp)\ Mod\, p$ /∗ Abstand gemäß (2.30) ∗/
(9) $\quad\quad a_i := A_{min} + r_i$ /∗ Stelle gemäß (2.31) ∗/
(10) $\quad\quad$**for** $(a := a_i; a \leq A_{max}; a := a+p)$ **do**
(11) $\quad\quad\quad$**if** $(H_b[a]=1)$ **then** /∗ (2.18) und (2.19) erfüllt? ∗/
(12) $\quad\quad\quad\quad$**while** $(p \mid S_b[a])$ **do**
(13) $\quad\quad\quad\quad\quad S_b[a] := S_b[a]/p$ /∗ Herausdividieren der maximalen Potenz von p ∗/
(14) $\quad\quad\quad\quad$**od**
(15) $\quad\quad\quad$**fi**
(16) $\quad\quad$**od**
(17) $\quad$**else**

Fortsetzung auf Seite 29

```
        if (b|p) then                              /* Teilt (p,∞)? */
          for (a := A_min; a ≤ A_max; a++)
              do
            if (H_b[a]=1) then                     /* (2.18) und (2.19) erfüllt? */
              while (p | S_b[a]) do
                S_b[a] := S_b[a]/p                 /* Herausdividieren der maxi-
                                                      malen Potenz von p */
              od
            fi
          od
        fi
      fi
    od
```

PHASE 3: AUSWERTUNG DER SIEBPHASE

```
    for (a := A_min; a ≤ A_max; a++) do
      if (S_b[a] ≠ 1) then                         /* (2.20) nicht erfüllt? */
        H̃_b[a] := 0                                /* (a,b) ausstreichen */
    od
    return (H̃_b)
```

Beispiel 2.19 (Fortsetzung von Beispiel 2.15)
Es seien wiederum $n = 391$ und $f(x) = x^3 + x^2 - 1$ sowie $FB_A = \{\pi_{5,3},\ \pi_{7,3},\ \pi_{11,2}\}$. Die folgende Tabelle zeigt die Änderung der Einträge im Siebarray bei der Verwendung des ALGSIEB–Algorithmus für $b = 3$. Mit x werden dabei die Stellen gekennzeichnet, an denen das aktuelle Faktorbasiselement zwar teilen würde, die aber im GGTSIEB oder RATSIEB schon ausgestrichen wurden.

a	-5	-4	-3	-2	-1	0	1	2	3	4	5
H_3	1	0	0	0	1	0	1	0	0	1	1
$S_3[a]$	173	x	x	x	23	x	25	x	x	43	77
$\pi_{5,3}$		x					1				
$\pi_{7,3}$				x							11
$\pi_{11,2}$											1
$\Rightarrow \tilde{H}_3$	0	0	0	0	0	0	1	0	0	0	1

Nach der Anwendung der drei gerade beschriebenen Siebalgorithmen auf eine Zeile im Rechteck AB lassen sich die guten Paare (a, b) leicht an ihrem Eintrag im Bitfeld $H_b[a]$, das der ALGSIEB–Algorithmus ausgegeben hat, erkennen. $H_b[a]$ ist nämlich genau dann 1, wenn (a, b) ein gutes Paar ist. Die zu konstruierende Menge T läßt sich somit durch folgendes Verfahren bestimmen.

Algorithmus 2.20 (Relationensuche)

Relationensuche

EINGABE: A_{min}, A_{max}, B_{max}, FB_R, FB_A
AUSGABE: Menge T von guten Paaren gemäß Definition 2.7

```
 (1)  T = ∅                                            /* Initialisierung */
 (2)  for (b := 1; b ≤ B_max; b ++) do
 (3)     H_b := GGTSIEB (b, A_min, A_max)              /* teste (2.18) */
 (4)     H_b := RATSIEB (b, A_min, A_max, H_b, FB_R)   /* teste (2.19) */
 (5)     H_b := ALGSIEB (b, A_min, A_max, H_b, FB_A)   /* teste (2.20) */
 (6)     for (a := A_min; a ≤ A_max; a ++) do
 (7)        if (H_b[a] = 1) then                       /* Ist (a, b) ein
                                                          gutes Paar? */
 (8)           T := T ∪ {(a, b)}                       /* erweitere T */
 (9)     od
(10)  od
(11)  return (T)
```

Beispiel 2.21 (Fortsetzung von Beispiel 2.19)
Die Relationensuche mit den Parametern $B_{max} = 5$, $A_{min} = -A_{max} = -5$ und den Faktorbasen $FB_R = \{2, 3, 5, 7, 11, 13\}$ und $FB_A = \{\pi_{5,3}, \pi_{7,3}, \pi_{11,2}\}$ liefert 13 gute Paare:

(a,b)	$a+bm$	$N(a,b)$	(a,b)	$a+bm$	$N(a,b)$
$(-3,1)$	4	-35	$(1,2)$	15	7
$(-3,4)$	25	1	$(1,3)$	22	25
$(-2,1)$	5	-11	$(1,5)$	36	121
$(-1,1)$	6	-1	$(2,1)$	9	5
$(-1,2)$	13	5	$(5,3)$	26	77
$(0,1)$	7	1	$(4,1)$	11	49
$(1,1)$	8	1			

Sind genügend viele Paare in T zusammengefaßt, so läßt sich daraus die Teilmenge S berechnen, die zur quadratischen Kongruenz (2.1) führt. Dazu verwendet man sogenannte Exponentenvektoren. Auf diese werde ich im nächsten Abschnitt genauer eingehen.

2.3.2 Quadratkonstruktion durch Exponentenvektoren

In diesem Teil der Arbeit möchte ich erläutern, wie man aus der in Kapitel 2.3.1 gebildeten Menge T guter Paare die Teilmenge S von Paaren herausfiltert, die die

Bedingungen

$$(2.7) \prod_{(a,b)\in S} (a+bm) = \tilde{x}^2,\ \tilde{x} \in \mathbb{Z}$$

und

$$(2.8) \prod_{(a,b)\in S} (a+b\rho) = \tilde{\delta}^2,\ \tilde{\delta} \in \mathbb{Q}(\rho)$$

bzw.

$$(2.9) \prod_{(a,b)\in S} (f_D \cdot a + b\omega) = \delta^2 \in \mathbb{Z}[\omega]$$

erfüllen.
Dazu möchte ich zuerst folgende Vorbetrachtung machen. Gegeben sei eine Zahl z aus $\mathbb{Z}$. Wie kann man entscheiden, ob mit z ein Quadrat vorliegt oder nicht? Eine notwendige Bedingung für die Quadrateigenschaft ist, daß alle in der Faktorisierung von z vorkommenden Primzahlen in gerader Potenz auftreten. Hinreichend ist diese Bedingung allerdings nicht, da sie die möglichen, in z enthaltenen Grundeinheiten unberücksichtigt läßt. In dieser Hinsicht gilt für Quadrate in $\mathbb{Z}$, daß sie positives Vorzeichen besitzen.
Aus dieser Vorbetrachtung heraus ergibt sich für unser Problem der Konstruktion von S, daß auf der einen Seite alle im Produkt der $a+bm$ auftretenden Primzahlen mit geradem Exponenten vorkommen müssen, und daß das Produkt der $a+bm$ positiv sein muß. Diese letzte Bedingung wird in der Praxis automatisch erfüllt sein, da m sehr viel größer als $|A_{min}|$ ist und nur positive b betrachtet werden. So bleibt also nur noch zu gewährleisten, daß alle Exponenten der Primzahlen gerade sind.
Ähnliche Betrachtungsweisen für die algebraische Seite führen zu Lemma 2.17 von Seite 28. Daraus folgt als notwendige Voraussetzung, daß

$$\sum_{(a,b)\in S} f_{(p,cp)}(a+b\rho) \equiv 0 \pmod 2$$

sein muß.
Definiert man für $\mathbb{K} = \mathbb{Q}(\rho)$

$$V_A = \{x \in \mathbb{K}^* \mid l_\pi(x) = 0 \text{ für alle Primideale } \pi \text{ von } A\},$$

bei l_π wie in Lemma 2.16, so gilt natürlich

$$\mathbb{K}^{*2} \subset V_A.$$

Weiter ist $V_A/\mathbb{K}^{*2}$ ein Vektorraum über $\mathbb{F}_2$ mit Dimension (siehe [BuLePo93, Kapitel 12.7])

$$dim_{\mathbb{F}_2} V_A/\mathbb{K}^{*2} \leq c \cdot logn$$

für eine Konstante c. Diese Tatsache kann ausgenutzt werden, um die Menge S wie gewünscht zu berechnen. Das Verfahren basiert auf dem folgenden Satz ([BuLePo93, Prop. 8.3]), der auf die Idee der *quadratischen Integer* von L. M. Adleman ([Ad91]) zurückgeht.

Satz 2.22
Sei S eine Menge von teilerfremden Paaren (a, b) mit der Eigenschaft, daß

$$\prod_{(a,b)\in S} (a + b\rho)$$

ein Quadrat in $\mathbb{Q}(\rho)$ ist. Sei weiter $(q, cq) \in R$ (siehe (2.16)) mit $cq \neq \infty$,

$$a + b \cdot cq \not\equiv 0 \pmod q \text{ für alle } (a, b) \in S \quad \text{und}$$

$$f'(cq) \not\equiv 0 \pmod q,$$

dann gilt

$$\prod_{(a,b)\in S} \left(\frac{a + b \cdot cq}{q}\right) = 1, \tag{2.33}$$

wobei $\left(\frac{x}{q}\right)$ das Legendresymbol (siehe Anhang A.3) bezeichnet. □

Setzt man für $FB_Q = \{(q_i, cq_i) \mid (q_i, cq_i) \text{ wie in Satz 2.22}\}$ und $|FB_Q| = B3 \in \mathbb{N}$

$$g_i^{(a,b)} := \begin{cases} 0 & \text{wenn } \left(\frac{a+b\cdot cq_i}{q}\right) = 1 \\ 1 & \text{wenn } \left(\frac{a+b\cdot cq_i}{q}\right) = -1, \end{cases} \tag{2.34}$$

so gilt aufgrund der Multiplikativität des Legendresymbols die Gleichung

$$\prod_{(a,b)\in S} \left(\frac{a + b \cdot cq_i}{q_i}\right) = 1$$

genau dann, wenn

$$\sum_{(a,b)\in S} g_i^{(a,b)} \equiv 0 \pmod 2$$

ist.
An den bisherigen Betrachtungen in diesem Kapitel kann man erkennen, daß zur Konstruktion von Quadraten in $\mathbb{Z}$ bzw. $\mathbb{Q}(\rho)$ die Exponenten der Faktorbasiselemente in den Zerlegungen von $a + bm$ bzw. $N(a, b)$ sowie die gerade definierten Zahlen $g_i^{(a,b)}$ eine wesentliche Rolle spielen. Deshalb definiere ich den Begriff Exponentenvektor folgendermaßen:

Definition 2.23 (Exponentenvektor)
Sei $(a, b) \in \mathbb{Z}^2$ mit $\mathrm{ggT}(a, b) = 1$,

$$a + bm = \prod_{p_i \in FB_R} p_i^{e_i^{(a,b)}}, e_i^{(a,b)} \in \mathbb{Z}$$

$$|N(a, b)| = \prod_{(p_i, cp_i)\in FB_A} p_i^{f_i^{(a,b)}}, f_i^{(a,b)} \in \mathbb{Z} \text{ und}$$

$$f_{B2+1}^{(a,b)} = \begin{cases} 1 & \text{wenn } N(a, b) < 0 \\ 0 & \text{sonst} \end{cases}$$

$$g_i^{(a,b)} \text{ wie in (2.34)},$$

dann heißt der Vektor $\vec{e}_{(a,b)} \in \mathbb{Z}^{B1+B2+B3+2}$

$$\vec{e}_{(a,b)} = \left(e_1^{(a,b)}, \ldots, e_{B1}^{(a,b)}, f_1^{(a,b)}, \ldots, f_{B2+1}^{(a,b)}, g_1^{(a,b)}, \ldots, g_{B3}^{(a,b)}, 1\right) \tag{2.35}$$

Exponentenvektor *von* (a, b). □

Bemerkung 2.24 (Exponentenvektor einer Menge von Paaren)
Mit $\vec{e}_M$ werde ich im folgenden den Exponentenvektor einer Menge M von Paaren (a, b) bezeichnen. Dieser sei wie folgt definiert:

$$\vec{e}_M := \sum_{(a,b)\in M} \vec{e}_{(a,b)}. \tag{2.36}$$

Damit gilt erstens

$$\prod_{(a,b)\in M} (a + bm) = \prod_{p_i \in FB_R} p_i^{\sum_{(a,b)\in M} e_i^{(a,b)}},$$

und zweitens

$$\prod_{(a,b)\in M} N(a, b) = (-1)^{\sum_{(a,b)\in M} f_{B2+1}^{(a,b)}} \cdot \prod_{(p_i, cp_i)\in FB_A} p_i^{\sum_{(a,b)\in M} f_i^{(a,b)}}.$$

□

Nach den obigen Betrachtungen müssen also zur Erfüllung der Quadratbedingungen (2.7) und (2.8) notwendigerweise alle Einträge des Exponentenvektors $\vec{e}_S$ gerade sein, wobei der letzte Eintrag gerade dafür zuständig ist, daß $|S|$ gerade wird. Die Umkehrung gilt leider nicht. Wählt man aber die Elemente von FB_Q zufällig und in genügend großer Anzahl, so kann man mit hoher Wahrscheinlichkeit davon ausgehen, daß $\prod_{(a,b)\in S} (a + b\rho)$ auch wirklich ein Quadrat ist. In [BuLePo93, Kapitel 12.7] wird vorgeschlagen,

$$B3 = [((c + 2)/\log 2) \log n]$$

zu wählen. Die Konstante c ist allerdings nicht bekannt. Setzt man sie aber zu null, so ergibt sich für $n \approx 10^{75}$

$$B3 = 498.$$

In der Praxis reichen allerdings wesentlich kleinere Werte, bis zu 100 (siehe Kapitel 7), aus.
Die Berechnung der Menge S aus der Menge T entspricht somit der Auswahl der Exponentenvektoren, deren Summe nur gerade Einträge hat. Da nur die Frage, welche Parität die Einträge haben, von Interesse ist, können wir uns auf die Betrachtung der Exponentenvektoren modulo 2 zurückziehen. Deshalb setzen wir

$$\begin{aligned}\underline{e}^{(a,b)} := \Big(& e_1^{(a,b)} \pmod 2, \ldots, e_{B1}^{(a,b)} \pmod 2,\\ & f_1^{(a,b)} \pmod 2, \ldots, f_{B2+1}^{(a,b)} \pmod 2,\\ & g_1^{(a,b)} \pmod 2, \ldots, g_{B3}^{(a,b)} \pmod 2\\ & 1 \pmod 2\Big).\end{aligned}$$

Findet man eine nichttriviale Linearkombination dieser Exponentenvektoren, so daß

$$\sum_{(a,b)\in T} \lambda(a,b)\cdot \underline{e}^{(a,b)} = \underline{0}$$

mit $\lambda(a,b)\in\{0,1\}$, dann ergibt sich S aus

$$S = \{(a,b)\in T \,|\, \lambda(a,b)=1\}.$$

Diese Linearkombination läßt sich durch Lösen des homogenen linearen Gleichungssystems

$$A\cdot\underline{\lambda} = \underline{0} \tag{2.37}$$

über $\mathbb{F}_2$ berechnen, wobei die Spalten von A, der sogenannten *Relationenmatrix*, durch die Vektoren $\underline{e}^{(a,b)}$ gebildet werden. Dieses Gleichungssystem hat mit Sicherheit dann eine nichttriviale Lösung, wenn A aus mehr Zeilen als Spalten besteht, also mindestens drei gute Paare mehr in T sind als Faktorbasiselemente in den drei Faktorbasen FB_R, FB_A und FB_Q zusammen.
Die Matrix A des Systems ist dabei dünnbesetzt, da für ein festes Paar (a,b) nur wenige Faktorbasiselemente in den Faktorisierungen von $a+bm$ bzw. $N(a,b)$ vorkommen.

Beispiel 2.25 (dünnbesetzte Matrix)
Bei der Faktorisierung von

$$Z_{50} = 25687932972608082604877595262293865434242386373017$$

unter Verwendung von Faktorbasen der Größen

$$|FB_R| = 2500, \quad |FB_A| = 15000, \quad |FB_Q| = 10$$

wurden 18026 Relationen in die Matrix aufgenommen. Von den etwa 315 Millionen Einträgen der Matrix waren 500583 ungleich Null, was durchschnittlich 27.7 Einträge pro Relation bedeutet. Dies heißt, daß im Durchschnitt nur 0.16% der Einträge pro Relation ungleich 0 sind. □

Zum Lösen dieser dünnbesetzen Systeme über endlichen Körpern gibt es spezielle, effiziente Algorithmen. Besonders zu nennen ist in diesem Zusammenhang die *Structured Gauss Methode* ([LaOd90]), mit deren Hilfe in den letzten Jahren weltweit viele Faktorisierungen fertiggestellt wurden. Asymptotisch bessere Laufzeiten weisen die *Block Lanczos Methode* ([Cp93]) und der *Block Wiedemann Algorithmus* ([Cp94]) auf. Deren Erforschung ist allerdings derzeit noch nicht weit fortgeschritten. So ist noch unklar, bei welcher Größe des Gleichungssystems welcher Algorithmus am effizientesten arbeitet.

Beispiel 2.26 (Fortsetzung von Beispiel 2.21)
Den Relationen entspricht die folgende Matrix A, wobei $FB_Q = \{(17,7), (19,16)\}$ gewählt wurde. Die letzte Zeile enthält nur Einsen. Sie ist dafür zuständig, daß die

Anzahl der Paare in der Menge S gerade ist.

$$A = \begin{pmatrix} 2 & 0 & 0 & 1 & 0 & 0 & 3 & 0 & 1 & 2 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 2 & 2 & 0 & 0 \\ 0 & 2 & 1 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 2 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 2 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 2 & 0 & 1 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \end{pmatrix}$$

Es gibt drei nichttriviale Lösungen für das Gleichungssystem $A \cdot x = 0$ modulo 2:

$$\begin{aligned} x_1 &= (0,0,1,1,1,0,0,1,0,0,1,1,0) \\ x_2 &= (0,1,0,0,0,0,0,0,0,1,0,0,0) \\ x_3 &= (0,0,0,0,0,0,1,0,1,1,0,0,1) \end{aligned}$$

□

Ein weiteres Problem ergibt sich durch den Übergang von $\mathbb{Q}(\rho)$ nach $\mathbb{Z}[\omega]$. Hat man nämlich die Menge S so generiert, daß

$$\prod_{(a,b)\in S} (a + b\rho) = \tilde{\delta}^2 \in \mathbb{Q}(\rho),$$

also

$$\prod_{(a,b)\in S} (f_D \cdot a + b\omega) = \eta^2 \in \mathbb{Z}[\omega],$$

so muß η zwar in $\mathcal{O}_\omega$ liegen, nicht aber notwendigerweise auch in $\mathbb{Z}[\omega]$.
Dieses Problem tritt nur auf, wenn $\mathbb{Z}[\omega] \neq \mathcal{O}_\omega$ ist, läßt sich dann aber leicht umgehen. Nach [Ws76](Prop. 3-7.14) gilt für $\alpha \in \mathcal{O}_\omega$:

$$g'(\omega) \cdot \alpha \in \mathbb{Z}[\omega]. \tag{2.38}$$

Hieraus folgt, daß

$$g'(\omega)^2 \cdot \prod_{(a,b)\in S} (f_D \cdot a + b\omega) = g'(\omega)^2 \cdot \eta^2 = \underbrace{(g'(\omega) \cdot \eta)}_{\in \mathbb{Z}[\omega]}^2 =: \delta^2 \tag{2.39}$$

ein Quadrat in $\mathbb{Z}[\omega]$ ist, wobei in diesem Fall δ selbst in $\mathbb{Z}[\omega]$ liegen muß.

Bemerkung 2.27 (Änderung in der quadratischen Kongruenz)
Multipliziert man $g'(\omega)^2$ an das algebraische Quadrat heran, so erhält man nicht mehr die quadratische Kongruenz

$$x^2 = f_D^{|S|} \cdot \prod_{(a,b)\in S} (a+bm) \equiv \psi\left(\prod_{(a,b)\in S} (f_D \cdot a + b\omega)\right) = y^2 \pmod{n},$$

sondern

$$\begin{aligned} \tilde{X}^2 &:= g'(f_D \cdot m)^2 \cdot {f_D}^{|S|} \cdot \prod_{(a,b)\in S} (a+bm) \\ &\equiv \psi\left(g'(\omega)^2 \cdot \prod_{(a,b)\in S} (f_D \cdot a + b\omega)\right) \pmod{n} \qquad (2.40) \\ &= \tilde{Y}^2 \pmod{n}. \end{aligned}$$

Ist $\mathrm{ggT}(g'(f_D \cdot m), n) = 1$, so hat diese neue Kongruenz eine ebenso gute Chance, zur Faktorisierung von n zu führen, wie die alte. Sind dagegen $g'(f_D \cdot m)$ und n nicht teilerfremd, so hat man damit einen Teiler von n gefunden. □

2.4 Berechnung der Quadratwurzeln

Nachdem die beiden Quadrate

$$g'(f_D \cdot m)^2 \cdot {f_D}^{|S|} \cdot \prod_{(a,b)\in S} (a+bm) \text{ in } \mathbb{Z} \qquad \text{und}$$

$$g'(\omega)^2 \cdot \prod_{(a,b)\in S} (f_D \cdot a + b\omega) \text{ in } \mathbb{Z}[\omega]$$

konstruiert wurden, müssen im nächsten Schritt, Schritt (3) des NFS Basisalgorithmus, ihre Quadratwurzeln berechnet werden.
Die rationale Wurzel läßt sich sehr leicht bestimmen, denn aufgrund der Zerlegung der $a + bm$ über der rationalen Faktorbasis gilt die Gleichheit

$$x^2 = g'(f_D \cdot m)^2 \cdot {f_D}^{|S|} \cdot \prod_{(a,b)\in S} (a+bm) = g'(f_D \cdot m)^2 \cdot {f_D}^{|S|} \cdot \prod_{p_i \in FB_R} p_i^{\sum_{(a,b)\in S} e_i^{(a,b)}}.$$

Da $g'(f_D \cdot m)^2 \cdot {f_D}^{|S|} \cdot \prod_{(a,b)\in S} (a+bm)$ ein Quadrat in $\mathbb{Z}$ und $|S|$ nach Konstruktion gerade ist, muß $\sum_{(a,b)\in S} e_i^{(a,b)}$ für jedes $i \in \{1, \ldots, B1\}$ gerade sein. Also ergibt sich für die Wurzel

$$x = g'(f_D \cdot m)^2 \cdot {f_D}^{|S|/2} \cdot \prod_{p_i \in FB_R} p_i^{\frac{1}{2} \cdot \sum_{(a,b)\in S} e_i^{(a,b)}}. \qquad (2.41)$$

Dieses Produkt läßt sich schnell mit dem Algorithmus FASTEXPO von Seite 161 bestimmen. Da wir das Resultat nach (2.1) nur modulo n benötigen, kann die gesamte Berechnung dabei modulo n durchgeführt werden.

Die Berechnung der Wurzel δ aus dem algebraischen Quadrat

$$\gamma = g'(\omega)^2 \cdot \prod_{(a,b)\in S} (f_D \cdot a + b\omega) \tag{2.42}$$

erweist sich als ungleich schwieriger.
Eine Möglichkeit besteht in der Faktorisierung des Polynoms

$$X^2 - \gamma \in \mathbb{Q}(\omega)[X]$$

über dem algebraischen Zahlkörpers $\mathbb{Q}(\omega)$. Verfahren, die diese Faktorisierung berechnen, findet man unter anderem in den Arbeiten [Wa76] oder [Wr76].

Beispiel 2.28 (Fortsetzung von Beispiel 2.26)
Die erste Lösung wird von den Paaren in

$$S = \{(-2,1), (-1,1), (-1,2), (1,2), (2,1), (5,3)\}$$

gebildet. Da in diesem Beispiel $\mathcal{O}_\omega = \mathbb{Z}[\omega]$ gilt, können wir auf die Erweiterung mit $g'(\omega)^2$ bzw. $g'(f_D \cdot m)^2$ verzichten. Auf der einen Seite gilt nun

$$1^{|S|} \cdot \prod_{(a,b)\in S} (a + bm) = 1368900 = 1170^2 \equiv (-3)^2 \pmod{391}$$

und auf der anderen

$$\prod_{(a,b)\in S} (1 \cdot a + b\omega) = 48\omega^2 - 59\omega + 25 = \left(\omega^2 - 5\omega + 6\right)^2.$$

Hieraus ergibt sich die quadratische Kongruenz

$$(-3)^2 \equiv \left(\psi\left(\omega^2 - 5\omega + 6\right)\right)^2 \equiv 20^2 \pmod{391},$$

die zum nichttrivialen Teiler $23 = \mathrm{ggT}(-3 - 20,\ 391)$ von 391 führt. □

Die Berechnung der Wurzel aus dem algebraischen Quadrat mit den oben angegebenen Verfahren ist allerdings wenig praktikabel, da die Koeffizienten des algebraischen Quadrates aus mehreren Millionen Dezimalstellen bestehen können.

Beispiel 2.29 (Größe des Quadrates)
Schon bei der Faktorisierung einer 80–stelligen Zahl treten riesige Koeffizienten auf. So setzt sich z.B. eine Lösung aus etwa $3.4 \cdot 10^5$ Paaren zusammen, die ein Quadrat ergeben, dessen Koeffizienten etwa 5.97 Millionen Dezimalstellen aufweisen. □

Einen effizienten und in der Praxis einsetzbaren Algorithmus zur Berechnung der algebraischen Quadratwurzel werde ich in Kapitel 5 vorstellen.

2.5 Die Laufzeit des *Number Field Sieve*

In diesem Abschnitt möchte ich auf die Laufzeit des NFS Algorithmus eingehen. Dazu werde ich zuerst eine abkürzende Schreibweise einführen.

Definition 2.30 (L_x)
Für u, $v \in \mathbb{R} > 0$ *bezeichne* $L_x[u, v]$

$$L_x[u, v] := \exp\left(v \cdot (\log x)^u \cdot (\log\log x)^{(1-u)}\right). \tag{2.43}$$

□

J.P. Buhler, H.W. Lenstra und C. Pomerance führten in ihrer Arbeit [BuLePo93, Kapitel 11] eine heuristische Laufzeitanalyse für das NFS durch. Sie zeigten, daß unter Verwendung eines Polynomes vom Grade

$$D = \left(3^{1/3} + o(1)\right) \cdot (\log n / \log\log n)^{1/3} \tag{2.44}$$

das NFS zum Faktorisieren einer Zahl n eine Laufzeit von

$$L_n\left[\frac{1}{3}, \left({}^{64}\!/\!_9\right)^{1/3} + o(1)\right] \tag{2.45}$$

aufweist.
Um diese Laufzeit besser einorden zu können, möchte ich folgende Bemerkung anfügen.

Bemerkung 2.31 (Einordnung der Laufzeit)
Betrachtet man für u in der Formel (2.43) aus Definition 2.30 nur Werte

$$0 \leq u \leq 1,$$

so ergeben sich als Extrema

u=0: $L_x[0, v] = exp(v \cdot \log\log(x))$

u=1: $L_x[1, v] = exp(v \cdot \log(x))$.

Der erste Ausdruck ist polynomiell in der Länge von x, der zweite dagegen exponentiell. □

Die Laufzeit des NFS liegt somit zwischen polynomiell und exponentiell in der Länge der zu faktorisierenden Zahl, ist also subexponentiell. Das zweite große allgemeine Faktorisierungsverfahren, das Quadratische Sieb (siehe [DDLM93]), weist mit

$$L_n\left[\frac{1}{2}, 1 + o(1)\right]$$

ebenfalls eine subexponentielle Laufzeit auf, die wegen der größeren zweiten Komponente allerdings asymptotisch schlechter als die des NFS ist. Ersetzt man in den Laufzeiten den $o(1)$–Term durch 0, so befindet sich der Cross–Over–Point, also die

Stelle, an der das NFS Verfahren schneller wird als das Quadratische Sieb, bei 125 Dezimalstellen.
In der Praxis hängt der Cross–Over–Point natürlich wesentlich von der jeweiligen Implementierung ab. Arjen Lenstra ([Le94]), der unter anderem an den Rekordfaktorisierungen mit dem Quadratischen Sieb beteiligt war, ist es im Sommer 1994 gelungen, im Bereich von 116 Dezimalstellen schneller mit dem NFS zu faktorisieren, als dies mit irgendeiner bekannten Implementierung des Quadratischen Siebes bisher gelungen ist.

Die Verbesserungen und Varianten des NFS–Grundalgorithmus, die ich in den nächsten Kapiteln vorstellen werde, können die heuristische Laufzeit zwar nicht verbessern, liefern aber in der Praxis enorme Laufzeitverbesserungen. Dabei werde ich insbesondere auf die drei Probleme

a.) die Berechnung des Polynoms $f(x)$ und dessen Wurzel in $\mathbb{Z}/n\mathbb{Z}$

b.) die Berechnung der Menge T von guten Paaren

c.) die Berechnung der Quadratwurzeln

eingehen.

Kapitel 3

Die Siebphase in der Praxis

In diesem Kapitel werde ich Varianten zu den Siebalgorithmen RATSIEB und ALGSIEB diskutieren, die zwar keine Verbesserung der asymptotischen Laufzeit darstellen, aber dennoch in der Praxis große Laufzeiteinsparungen bewirken.
Das Kapitel gliedert sich in mehrere Abschnitte. Zuerst werde ich Gemeinsamkeiten zwischen der rationalen und algebraischen Seite des Verfahrens herausarbeiten. In den beiden folgenden Abschnitten werden einige schon aus anderen Faktorisierungsalgorithmen bekannte Strategien zur Beschleunigung der Siebphase vorgestellt.
Im vierten Teil werde ich eine neue Erweiterung der Large Prime Variante namens Quadruple Large Prime Variante beschreiben und vor allem darauf eingehen, wie man diese effizient im NFS nutzen kann. In diesem Rahmen werden mehrere neue Algorithmen vorgestellt, die zum einen die Siebphase aber auch zum anderen das Kombinieren der Relationen mit Large Primes betreffen.
Anschließend werde ich das Lattice Sieve, eine Alternative zum normalen Siebvorgang, diskutieren. Den Abschluß dieses Kapitels bilden Betrachtungen über die Möglichkeit, die Siebphase auf mehreren Rechnern verteilt durchzuführen.

Bevor ich mit der Beschreibung von Details beginne, möchte ich noch einige grundsätzliche Bemerkungen über das Prinzip des Siebens im Zusammenhang mit dem Faktorisieren machen:
Effizient zu faktorisieren bedeutet, möglichst schnell einen nichttrivialen Teiler einer zusammengesetzten Zahl zu finden. Im NFS entspricht Effizienz, wie wir noch sehen werden, größtenteils dem schnellen Finden von genügend vielen Relationen. Dabei darf man, um Laufzeit einzusparen, gute Paare in einem vorgegebenen Bereich verpassen, muß allerdings darauf achten, daß deren Anzahl nicht zu groß wird, so daß immer noch genügend viele Relationen erzeugt werden.

3.1 Ähnlichkeiten zwischen rationaler und algebraischer Seite

Bisher habe ich immer streng zwischen der rationalen Seite, der Faktorbasis FB_R und den zu zerlegenden Zahlen $|a + b\,m|$ einerseits, und der algebraischen Seite, der Faktorbasis FB_A und den Werten $N(a, b)$ andererseits unterschieden. Dennoch gibt es zwischen diesen beiden Seiten viele Gemeinsamkeiten. So läßt sich die rationale Faktorbasis wie die algebraische als Menge von Paaren (p, cp) darstellen. Für diese Paare gilt

$$(p, cp) = (p, m \bmod p).$$

Man sieht leicht, daß die cp analog zur algebraischen Faktorbasis gerade den Nullstellen des Polynoms $r(x) = x - m$ modulo p entsprechen. Ein Faktorbasiselement (p, cp) teilt $a + bm$ wegen

$$a + bm \equiv a + b \cdot cp \pmod{p}$$

genau dann, wenn

$$a + b \cdot cp \equiv 0 \pmod{p}$$

ist. Die im algebraischen Fall möglichen Paare (p, ∞) kommen im rationalen Fall allerdings nicht vor, da der höchste Koeffizient des rationalen Polynoms $r(x)$ gerade 1 ist.

Eine weitere Übereinstimmung zwischen der algebraischen und der rationalen Seite kann man feststellen, wenn man z.B. die Berechnung von $N(a, b)$ betrachtet. Für die $N(a, b)$ gilt nach (2.17)

$$|N(a,b)| = \left| f\left(\frac{-a}{b}\right) \cdot (-b)^D \right|.$$

Wendet man diese Formel analog auf das 'rationale' Polynom $r(x)$ an, so gilt

$$\left| r\left(\frac{-a}{b}\right) \cdot (-b) \right| = \left| \left(\frac{-a}{b} - m\right) \cdot (-b) \right| = |a + bm|.$$

Aus dieser Sichtweise heraus kann man z.B. die Algorithmen RATSIEB bzw. ALGSIEB zu einem umfassenderen Algorithmus zusammenfassen, der je nach Wahl der Parameter 'Polynom' und 'Faktorbasis' dem Algorithmus RATSIEB bzw. ALGSIEB entspricht. Im weiteren werde ich mich bei der Beschreibung der Algorithmen, wenn immer möglich, auf diese Betrachtungsweise zurückziehen. Dabei werde ich mit $N_h(a, b)$ für ein Polynom $h(x) \in \mathbb{Z}[x]$ gerade

$$h\left(-a/b\right) \cdot (-b)^{deg(h)} \tag{3.1}$$

bezeichnen.

3.2 Sieben mit Logarithmen

Die Algorithmen RATSIEB auf Seite 25 und ALGSIEB auf Seite 29 erfordern sehr viele Divisionen durch Faktorbasiselemente. Die meisten dieser Divisionen sind in dem Sinne unnötig, daß sie für Paare (a, b) durchgeführt werden, die später nicht als gutes Paar anerkannt werden. Eine wesentliche Beschleunigung der Siebphase erhält man durch Ersetzen eines großen Teils dieser teuren Divisionen durch billige Subtraktionen. Dies läßt sich dadurch erreichen, daß man

1. die Siebeinträge durch ihre Logarithmen, also durch $\log(|N_r(a,b)|)$ im rationalen Fall und $\log(|N_f(a,b)|)$ im algebraischen Fall, ersetzt;

2. an den Stellen, an denen das Faktorbasiselement p bzw. $\pi_{p,cp}$ teilt, $\log(p)$ vom Siebeintrag subtrahiert.

Nach Durchführung dieser Siebvariante entsprechen die Stellen im Siebarray, die 0 enthalten, den Paaren, für die $a + bm$ bzw. $|N(a,b)|$ total über der Faktorbasis zerfallen.
Diese Vorgehensweise birgt jedoch einige kleine Schwierigkeiten, die Gegenstand der nächsten Abschnitte sind.

Ungenauigkeit

Während in den beiden ursprünglichen Siebalgorithmen ausschließlich mit ganzen Zahlen gearbeitet wird, muß in der Variante des Siebens mit Logarithmen auf reelle Zahlen zurückgegriffen werden. Da diese im Rechner nicht genau dargestellt werden (z.B. werden vom Datentyp *double* nur 52 Bit zur Darstellung der Mantisse benutzt), sind auch die Rechenergebnisse ungenau. Dies bedeutet, daß, selbst wenn der entsprechende Wert über der Faktorbasis zerfällt, die Null am Ende des Siebes nicht genau erreicht werden muß. Zur Behebung dieses Defekts wird eine kleine Schranke namens KORRFAK gesetzt, bei deren Unterschreitung man davon ausgeht, daß die Null erreicht wurde.
Je nachdem, wie groß diese Schranke gewählt wird, ist es allerdings möglich, auch solche Paare als gut anzuerkennen, die in Wirklichkeit keine sind. Um diesen Fehler auszugleichen, müssen alle vom Siebalgorithmus akzeptierten, möglicherweise guten Paare auf Korrektheit überprüft werden. Dies geschieht mit Hilfe eines Probedivisionsalgorithmus. Für ein gegebenes Paar (a, b) wird damit überprüft, ob die Werte

$$a + bm \quad \text{bzw.} \quad |N(a,b)|$$

über den Faktorbasen zerfallen. Dabei wird nacheinander für alle Faktorbasiselemente untersucht, ob sie den entsprechenden Wert teilen. Wenn dem so ist, wird die größtmögliche Potenz herausdividiert. Gemäß den Betrachtungen in Kapitel 3.1 wird der Probedivisionsalgorithmus je nach Wahl der Parameter den rationalen bzw. algebraischen Wert zerlegen. Für den Test, ob ein vorliegendes Paar (a, b) gut ist, muß man dann zuerst

$$\text{zerfällt\_rational=PROBEDIVISION}(a,\ b,\ x - m,\ FB_R)$$

und, wenn dies TRUE ist,

$$\text{zerfällt\_algebraisch=PROBEDIVISION}(a,\ b,\ f(x),\ FB_A)$$

aufrufen.

Algorithmus 3.1 (Probedivision)

PROBEDIVISION

EINGABE: a, b, $h(x)$, FB
AUSGABE:

$$\begin{array}{rl} TRUE, & \text{wenn } |N_h(a,b)| \text{ über } FB \text{ zerfällt} \\ FALSE, & \text{sonst} \end{array}$$

```
Nab := |N_h(a, b)|                          /* Initialisierung */
foreach ((p, cp) ∈ FB) do                   /* für alle Faktorbasiselemente */
  if (cp ≠ ∞) then
    if (a + b · cp ≡ 0 (mod p))             /* Teilt (p, cp)? */
      then
      do
        Nab := Nab/p                        /* Herausdividieren der maxi- */
      while (p | Nab)                       /* malen Potenz von p */
    fi
  else
    if (p | b) then                         /* Teilt (p, ∞)? */
      do
        Nab := Nab/p                        /* Herausdividieren der maxi- */
      while (p | Nab)                       /* malen Potenz von p */
    fi
  fi
od
if (Nab ≠ 1) then                           /* N_h(a, b) zerfällt nicht */
  return (FALSE)                            /* über der Faktorbasis */
else
  return (TRUE)
```

In der Praxis empfiehlt sich, zum Abspeichern der Logarithmen anstelle eines Fließkomma–Typs, wie z.B. *double* (8 Bytes), den Datentyp *char* (1 Byte) zu verwenden. Der Grund dafür liegt eben im achtmal so großen Hauptspeicherplatzbedarf einer *double*–Variablen. Nimmt man z.B. ein Siebarray der Länge 10^6, so kann durch die Verwendung von *char* statt *double* etwa 7 Megabyte Hauptspeicher eingespart werden.
Da in einem *char* aber nur ganzzahlige Werte zwischen -128 und 127 abgespeichert werden können, muß eine recht grobe Approximation für den Logarithmus verwendet werden. In der Praxis kann man beispielsweise

$$LOG_p = \lfloor \log p + 0.5 \rfloor$$

entsprechend den Faktorbasiselementen (p, cp) verwenden, wobei mit log der natürliche Logarithmus bezeichnet wird. Etwas genauere Näherungen kann man erhalten, wenn man stattdessen einen Logarithmus zu einer kleineren Basis, z.B. 2 oder 1.6, wählt.

Die Schranke KORRFAK spielt für die Effizienz des Siebens mit Logarithmen eine wichtige Rolle. Setzt man sie zu niedrig, so verliert man viele Relationen, setzt man sie dagegen zu hoch, so muß für viele Paare, die sich schließlich als keine guten herausstellen, der Probedivisonsalgorithmus durchgeführt werden. Die Wahl des Korrekturfaktors wird aber auch noch durch weitere algorithmische Verbesserungen, die ich in späteren Kapiteln beschreiben werde, beeinflußt, so daß ich erst im Kapitel 6 darauf eingehen möchte, wie man KORRFAK in der Praxis wählen soll.

Potenzen werden nicht berücksichtigt

In den Schritten (10) bis (12) der Algorithmen RATSIEB und ALGSIEB werden höhere Potenzen der Faktorbasiselemente herausdividiert. Beim Sieben mit Logarithmen ist dies so nicht möglich, da man dem Siebeintrag i.a. nicht ansehen kann, ob und in welcher Potenz ein Faktorbasiselement teilt.
Damit man aber nicht auf alle Paare (a, b) verzichten muß, in deren Zerlegungen höhere Potenzen von Faktorbasiselementen vorkommen, muß wiederum an der KORRFAK–Schraube gedreht werden.

Verzicht auf kleine und spezielle Faktorbasiselemente

Ein großer Teil der Laufzeit des Siebens mit Logarithmen wird von den kleinen Faktorbasiselementen verursacht. So wird z.B. die Primzahl 2 an jeder zweiten Stelle den Siebeintrag teilen. Dies bedeutet, daß für ein Siebarray der Länge L $L/2$ Additionen benötigt werden, um die entsprechenden Stellen zu lokalisieren. Ebenso oft wird log(2) subtrahiert. Für die Siebeinträge bedeutet dies aber nur eine geringe Reduzierung. Für große Faktorbasiselemente benötigt man wesentlich weniger Additionen und hat zusätzlich den Vorteil, daß wesentlich größere Logarithmenwerte subtrahiert werden. Eine verbesserte Laufzeit läßt sich deshalb dadurch erreichen, daß man mit den kleinen Faktorbasiselementen, etwa den ersten 25, nicht siebt, wohl aber Probedivision durchführt. Einen weiteren kleinen Laufzeitvorteil kann man sich dadurch verschaffen, daß man die speziellen Faktorbasiselemente, d.h. die Elemente (p, ∞) ebenfalls während des Siebvorgangs unberücksichtigt läßt. Dies schadet aus zwei Gründen nichts. Erstens gibt es nur wenige dieser Faktorbasiselemente, und zweitens sind die meisten davon so klein, daß man mit ihnen sowieso nicht gesiebt hätte. Für die Laufzeit der Siebphase bringt dieses Vorgehen allerdings den Vorteil, daß weniger Fallunterscheidungen (im Programmcode sind dies if–Abfragen) gemacht werden müssen.
Dem durch den Verzicht auf einige Faktorbasiselemente begangenen Fehler beim Sieben mit Logarithmen kann wiederum durch geeignete Wahl des Korrekturfaktors KORRFAK begegnet werden.
Diese Variante des Siebens mit Logarithmen wird im folgenden Algorithmus zusammengefaßt.

Algorithmus 3.2 (Sieben mit Logarithmen)

LOGSIEB

EINGABE: b, A_{min}, A_{max}, H_b, FB, $h(x)$, KORRFAK
AUSGABE: Bitarray $\tilde{H}_b$ mit

$$\tilde{H}_b[a] = \begin{cases} 1, & \text{wenn } H_b[a] = 1 \text{ und } \log|N_h(a,b)| \\ & \text{sich bis auf KORRFAK durch die} \\ & \text{Faktorbasiselemente reduzierenläßt} \\ 0, & \text{sonst} \end{cases}$$

PHASE 1: INITIALISIERUNG

(1)	**for** $(a := A_{min}; a \le A_{max}; a{+}{+})$ **do**	
(2)	$\quad\tilde{H}_b[a] := H_b[a]$	/* Bitarray */
(3)	$\quad$**if** $(H_b[a] = 1)$ **then**	/* (2.18) und (2.19) erfüllt? */
(4)	$\quad\quad S_b[a] := \log(\lvert N_h(a,b)\rvert)$	/* Siebarray */
(5)	**od**	

PHASE 2: SIEBPHASE

(6)	**foreach** $((p, cp) \in FB)$ **do**	/* für alle Faktorbasiselemente (p, cp) */
(7)	$\quad r := -(A_{min} + b \cdot cp)\ Mod\, p$	/* Abstand gemäß (2.30) */
(8)	$\quad a_p := A_{min} + r$	/* Stelle gemäß (2.31) */
(9)	$\quad$**for** $(a := a_p; a \le A_{max}; a := a + p)$ **do**	
(10)	$\quad\quad$**if** $(H_b[a] = 1)$ **then**	/* weiterhin brauchbar ? */
(11)	$\quad\quad\quad S_b[a] := S_b[a] - \log p$	/* Logarithmus subtrahieren */
(12)	$\quad$**od**	
(13)	**od**	

PHASE 3: AUSWERTUNG DER SIEBPHASE

(14)	**for** $(a := A_{min}; a \le A_{max}; a{+}{+})$ **do**	
(15)	$\quad$**if** $(S_b[a] >$ KORRFAK$)$ **then**	/* Siebeintrag zu groß */
(16)	$\quad\quad\tilde{H}_b[a] := 0$	/* Streichen des Paares (a, b) */
(17)	**od**	
(18)	**return** $(\tilde{H}_b)$	

Beispiel 3.3 (Laufzeitvergleich)
Beim Sieben für eine 89–stellige Zahl mit den Parametern

$$\begin{aligned} |FB_R| &= 20000 \\ |FB_A| &= 60160 \\ A_{max} &= -A_{min} = 5000000 \end{aligned}$$

wurden für $b = 7500$ folgende Laufzeiten gemessen: Siebt man mit allen Faktorbasiselementen, so benötigt das Sieben mit Logarithmen 22.83 Sekunden, ohne die kleinsten 25 Faktorbasiselemente dagegen nur 13.33 Sekunden. Dies entspricht einer Laufzeitverbesserung von etwa 42%. Durch die Veränderung des Korrekturfaktors werden allerdings beim Sieben ohne die ersten Faktorbasiselemente einige Paare mehr als gut erkannt. Diese werden jedoch wieder vom Probedivisionsalgorithmus elimiert. Die dazu benötigte Zeit muß aber auch berücksichtigt werden, wenn man die Laufzeitverbesserung untersucht, die diese Variante bewirkt.
Insgesamt benötigt man durchschnittlich 2.95 Sekunden pro Relation, wenn man mit allen Faktorbasiselementen siebt, nur 2.66 Sekunden dagegen, wenn man auf die kleinen Elemente verzichtet. In diesem Beispiel führt diese Variante also immer noch zu einer Laufzeitminderung von etwa 10%. □

3.3 Die Large Prime Variante

3.3.1 Die Idee der Large Prime Variante

Analysiert man die Reste, die nach dem Sieben mit der rationalen Faktorbasis im RATSIEB–Algorithmus in den Siebeinträgen verbleiben, so kann man feststellen, daß diese oftmals nur eine Primzahl enthalten, die größer als die größte Primzahl in der rationalen Faktorbasis ist. Man hat also eine Zerlegung der Form

$$a + b \cdot m = \prod_{p_i \in FB_R} {p_i}^{e_i} \cdot lp, \tag{3.2}$$

wobei $lp \in \mathbb{P}$, $lp > p_{B1}$, gefunden. Die Zahl lp heißt in diesem Zusammenhang *rationale Large Prime*. Findet man ein weiteres Paar $(\tilde{a}, \tilde{b})$, so daß

$$\tilde{a} + \tilde{b} \cdot m = \prod_{p_i \in FB_R} {p_i}^{\tilde{e}_i} \cdot lp,$$

dann kann man beide Paare zum Aufstellen des Gleichungssystems modulo 2 nutzen, da alle Teiler von

$$(a + b \cdot m) \cdot (\tilde{a} + \tilde{b} \cdot m) = \prod_{p_i \in FB_R} {p_i}^{e_i + \tilde{e}_i} \cdot lp^2,$$

die nicht in der rationalen Faktorbasis liegen, in gerader Potenz vorkommen. Ich sage dazu, die Large Prime wurde zu einem Quadrat kombiniert oder die Paare (a, b) und $(\tilde{a}, \tilde{b})$ wurden kombiniert.
Um sicherzustellen, daß ein Rest größer 1 im Siebarray nicht zusammengesetzt ist, mußte man eigentlich einen Primzahltest durchführen. Dies läßt sich aber leicht

dadurch umgehen, daß man eine Obergrenze LPBOUND für die rationalen Large Primes einführt und diese auf einen Wert zwischen

$$p_{B1} < LPBOUND < {p_{B1}}^2$$

setzt. Dann ist nämlich ein Rest r im Siebarray mit

$$p_{B1} < r \leq LPBOUND \tag{3.3}$$

garantiert eine Primzahl, da während des Siebens durch alle Primzahlen kleiner als $\sqrt{LPBOUND}$ dividiert wurde.
Je größer man die Schranke LPBOUND wählt, umso mehr Paare (a, b) findet man und umso mehr Kombinationen der Large Primes zu Quadraten erhält man. Dem stehen auf der anderen Seite zwei negative Aspekte gegenüber. Erstens nimmt die Wahrscheinlichkeit, daß eine Large Prime in mehr als einer Zerlegung auftaucht, mit der Größe der Large Prime ab und zweitens führt eine Erhöhung von LPBOUND zu größeren Datenmengen, die man zu verwalten hat.

Analog zu der Vorgehensweise im rationalen Fall kann man auch algebraische Large Primes erlauben. Für ein Paar (a, b) werden dann also auch Zerlegungen

$$|N(a,b)| = \prod_{(p,cp)\in FB_A} p^{f(p,cp)} \cdot l\pi \tag{3.4}$$

akzeptiert. Auch in diesem Fall wird eine Obergrenze LPIBOUND für die algebraischen Large Primes eingeführt. Diese muß, um Primzahltests zu verhindern, im Bereich

$$p_{B2} < LPIBOUND < {p_{B2}}^2$$

liegen. Algebraische Large Primes $l\pi$, die akzeptiert werden, müssen dann eine Primzahl

$$l\pi \leq LPIBOUND$$

sein.
Durch das Akzeptieren sowohl von rationalen als auch von algebraischen Large Primes erhält man möglicherweise Paare (a, b), die zwei Large Primes besitzen. Deshalb nennt man dieses Verfahren auch *Double Large Prime Variante* (DLP).
War es bei einer Large Prime pro Paar noch einfach zu entscheiden, welche Paare miteinander kombiniert werden mußten, so ist dies bei zwei möglichen Large Primes pro Paar (a, b) wesentlich schwieriger, wie folgendes kleine Beispiel zeigt.

Beispiel 3.4 (Paare mit maximal zwei Large Primes)
Angenommen es seien die Paare wie in Tabelle 3.1 mit maximal zwei Large Primes gefunden worden.
Dann werden durch Verwendung der Paare $(a_1, b_1), \ldots, (a_8, b_8)$ ohne (a_6, b_6) die Large Primes zu Quadraten kombiniert, denn es gilt

$$\begin{aligned}
\prod_{\substack{i=1\\i\neq 6}}^{8}(a_i + b_i \cdot m) &= \prod_{p\in FB_R} p^{e_p} \cdot {lp_1}^2 \cdot {lp_2}^2 \cdot {lp_3}^2 \\
\prod_{\substack{i=1\\i\neq 6}}^{8}|N(a_i, b_i)| &= \prod_{(p,cp)\in FB_A} p^{f(p,cp)} \cdot {l\pi_1}^2 \cdot {l\pi_2}^2 \cdot {l\pi_3}^2
\end{aligned}$$

Tabelle 3.1: Relationen mit Large Primes

Paar	rationale Large Prime	algebraische Large Prime
(a_1, b_1)	lp_1	$l\pi_1$
(a_2, b_2)	lp_2	$l\pi_2$
(a_3, b_3)	lp_3	$l\pi_1$
(a_4, b_4)	lp_1	$l\pi_3$
(a_5, b_5)	–	$l\pi_2$
(a_6, b_6)	lp_4	$l\pi_1$
(a_7, b_7)	lp_3	–
(a_8, b_8)	lp_2	$l\pi_3$

□

Das Akzeptieren von Large Primes macht eine Neudefinition der Begriffe *gute Paare* bzw. *Relation* (siehe Definition 2.7) erforderlich.

Definition 3.5 (gute Paare, Relation (Version 2))
Ein Paar $(a, b) \in \mathbb{Z}^2$ *heißt* gutes Paar *oder* ***Relation****, wenn für gegebene Faktorbasen* FB_R *und* FB_A *sowie Schranken LPBOUND, LPIBOUND* $\in \mathbb{R}$ *folgende drei Bedingungen erfüllt sind:*

$$\mathrm{ggT}(a, b) = 1 \tag{3.5}$$

$$a + bm = \prod_{p \in FB_R} p^{e_p} \cdot lp,\; e_p \in \mathbb{N}_0 \tag{3.6}$$

wobei $lp = 1$ *oder* lp *Primzahl kleiner LPBOUND ist, die nicht in der rationalen Faktorbasis* FB_R *liegt.*

$$|N(a, b)| = \prod_{(p,cp) \in FB_A} p^{f_{(p,cp)}} \cdot l\pi,\; f_{(p,cp)} \in \mathbb{N}_0 \text{ wie in (2.28),} \tag{3.7}$$

wobei $l\pi = 1$ *oder* $l\pi$ *eine Primzahl kleiner LPIBOUND ist, die nicht in einem Paar der algebraischen Faktorbasis vorkommt.* □

Bemerkung 3.6 (Schreibweise)
Ein gutes Paar (a, b), heißt

Full Relation oder kurz **Full**, wenn $lp = 1$ und $l\pi = 1$,

Single Large Prime Relation oder kurz **Single**,

wenn $(lp = 1 \wedge\; l\pi \neq 1)$ oder $(lp \neq 1 \wedge\; l\pi = 1)$,

Double Large Prime Relation oder kurz **Double**,

wenn $lp \neq 1 \wedge \; l\pi \neq 1$. □

Wie wirkungsvoll die Verwendung von Large Primes in der Praxis ist, demonstriert das folgende Beispiel.

Beispiel 3.7 (Faktorisieren mit zwei Large Primes)
Zur Faktorisierung der 50–stelligen Zahl Z_50 (siehe Kapitel 7.5) wurden

$$FB_R = \{p \in \mathbb{P} \,|\, p \leq 22307\} \, (B1 = 2500) \text{ und}$$

$$FB_A = \{\pi_{p,cp} \,|\, p \leq 163819\} \, (B2 = 14707)$$

verwendet. Die Grenzen für die Large Primes wurden auf

$$\begin{array}{rcl} LPBOUND & = & 3 \cdot 10^5 \\ LPIBOUND & = & 1.5 \cdot 10^6 \end{array}$$

gesetzt. Die Verteilung der Relationen wird in Tabelle 3.2 dargestellt.

Tabelle 3.2: Relationenverteilung am Bsp. Z_{50}

Full	7982
Single	27633
Double	20314
insgesamt	55929

Die Relationen mit Large Primes konnten auf 10044 unabhängige Weisen so kombiniert werden, daß alle Large Primes nur noch in gerader Potenz vorkamen. Dies bedeutet, daß in der gleichen Siebzeit durch Berücksichtigung von Large Primes die Anzahl der Spalten des Gleichungssystems mehr als verdoppelt wurden. Dies entspricht einer Gesamtlaufzeitverkürzung um etwa 50%. □

Im nächsten Abschnitt werde ich ausführlich darauf eingehen, welche Paare (a, b) man nehmen muß, so daß alle Large Primes in den entsprechenden Zerlegungen in gerader Potenz vorkommen.

3.3.2 Kombinieren der Large Primes

Die Idee

In diesem Abschnitt werde ich einen Algorithmus vorstellen, der gute Paare (a, b) mit Large Primes so kombiniert, daß alle vorkommenden Large Primes in gerader

Potenz erscheinen. Dabei handelt es sich um eine Modifikation des Verfahrens von A. K. Lenstra und M. Manasse ([LeMa]).

Im weiteren möchte ich keinen Unterschied zwischen rationalen und algebraischen Large Primes machen. Da gemäß den Betrachtungen aus Kapitel 3.1 sich jede Large Prime als Paar (lp, lcp) mit $lcp \in \{0, \ldots, lp-1\}$ oder $lcp = \infty$ schreiben läßt, verwende ich diese Darstellung für die Large Primes mit der einzigen Ausnahme, daß ein Paar (lp, ∞) als (lp, lp) geschrieben wird. Um zu vermeiden, daß rationaler und algebraischer Large Prime dasselbe Paar zugeordnet wird, wird einer rationalen Large Prime (lp, lcp) der Wert $(lp, -lcp)$ zugedacht. Zur Vereinfachung werde ich im folgenden die Zahl 1 sowohl als rationale als auch als algebraische Large Prime ansehen und ihr das Paar $(1, 0) \in \mathbb{Z}^2$ zuordnen. Jede Relation hat damit genau eine rationale und eine algebraische Large Prime.
Die Idee des Algorithmus von Lenstra und Manasse besteht in der Übertragung obigen Problems auf das Problem, Zykel in einem ungerichteten Graphen zu finden. Der Graph $G = (V, E)$ besteht dabei aus der Knotenmenge

$$V = \Big\{(lp, lcp) \in \mathbb{Z}^2 \,|\, (lp, lcp) \text{ entspricht rationaler oder algebraischer Large Prime}\Big\}$$

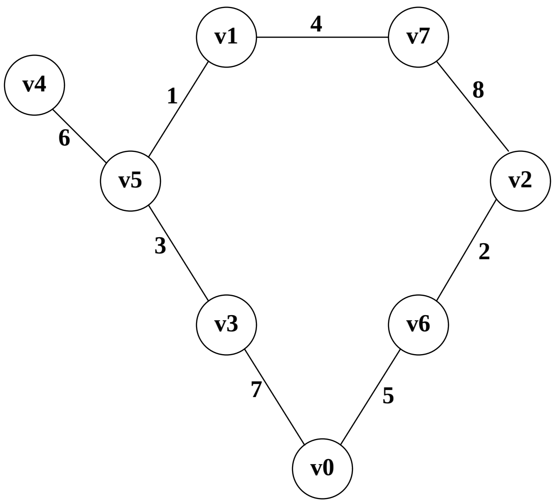
und der Kantenmenge

$$E \subseteq \{((lp_1, lcp_1), (lp_2, lcp_2)) \mid \begin{array}{l}\text{die erste Komponente entspricht der rationalen}\\ \text{und die zweite der algebraischen Large Prime}\\ \text{eines Paares } (a,b)\ \}\end{array}$$

Jedem Zykel in diesem Graphen entspricht dann eine Kombination der Relationen, in der alle Large Primes nur in gerader Potenz vorkommen.

Beispiel 3.8 (Zykel in einem Graphen)
Den entsprechenden Graphen zu Beispiel 3.4 zeigt die Graphik 3.1.
Zur Verdeutlichung beschrifte ich die Kanten des Graphen mit dem Index des dieser Kante entsprechenden Paares. Die 8 Knoten v_0 bis v_7 des Graphen entsprechen $(1, 0)$, lp_1, lp_2, lp_3, lp_4, $l\pi_1$, $l\pi_2$, $l\pi_3$.

Abbildung 3.1: Beispielgraph



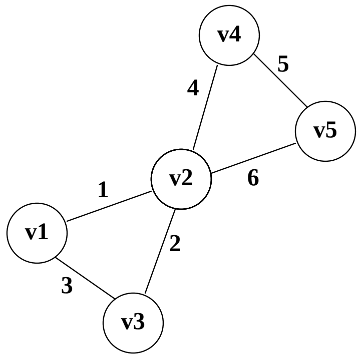
Wie man sieht, entsprechen dem Zykel im Graphen analog zu Beispiel 3.4 genau die Paare $(a_1, b_1) \ldots (a_8, b_8)$ mit Ausnahme von (a_6, b_6). □

Wir benötigen allerdings nicht alle Zykel des Graphen, sondern nur die unabhängigen, denn abhängige Zykel führen zu abhängigen Spalten des linearen Gleichungssystems (2.37) (siehe Seite 35), die aber nur zu Lösungen (x, y) der Quadratischen Kongruenz (2.1) von Seite 15 führen, die die Bedingung

$$(2.3) \quad x \not\equiv \pm y \pmod n$$

verletzen.
Ein Beispiel für abhängige Zykel zeigt der Graph aus Abbildung 3.2.

Abbildung 3.2: Graph mit abhängigen Zykel



Die 3 verschiedenen Zykel des Graphen sind

1. $v1 \longleftrightarrow v2 \longleftrightarrow v3 \longleftrightarrow v1$
2. $v2 \longleftrightarrow v4 \longleftrightarrow v5 \longleftrightarrow v2$ und
3. $v1 \longleftrightarrow v2 \longleftrightarrow v4 \longleftrightarrow v5 \longleftrightarrow v2 \longleftrightarrow v3 \longleftrightarrow v1$

Der dritte Zykel ist aber durch das Kombinieren der beiden ersten entstanden, ist also abhängig von ihnen.

Die Realisierung

Zur Berechnung der unabhängigen Zykel des Graphen wird in der Praxis den Kanten des Graphen eine Richtung gegeben, also ein gerichteter Graph verwendet. Die Zykel des entsprechenden ungerichteten Graphen werden dabei schon während des Aufbaus des Graphen lokalisiert und entfernt, so daß keine abhängigen Zykel entstehen können.
Um den Kanten des Graphen eindeutig eine Richtung zuordnen zu können, benötigt man die Definition einer Ordnung auf der Menge der Knoten.

Definition 3.9 (Ordnung der Knoten)
Für zwei Knoten $v_1 = (lp_1, lcp_1)$ und $v_2 = (lp_2, lcp_2)$ aus V heißt v_1 kleiner als v_2 ($v_1 < v_2$), wenn gilt

$$lp_1 < lp_2 \vee \ (lp_1 = lp_2 \wedge \ lcp_1 < lcp_2)$$

□

Bemerkung 3.10 (kleinster Knoten)

1. $(V, <)$ definiert eine totale Ordnung auf V.
2. Der kleinste Knoten gehört zur Pseudo Large Prime $(1, 0)$, da alle anderen Knoten in der ersten Komponente eine Primzahl enthalten. □

Der gerichtete Graph wird nun so aufgebaut, daß von jedem Knoten höchstens eine Kante ausgeht. Dabei wird gewährleistet, daß in jeder Zusammenhangskomponente des Graphen genau ein Knoten ohne ausgehende Kante existiert. Solche Knoten bezeichne ich mit *Wurzel*. Die Kanten des Graphen werden so gerichtet, daß, wenn man von einem beliebigen Knoten einer Zusammenhangskomponente aus über die Kanten des Graphen läuft, genau in der Wurzel der Komponente endet.
Die Initialisierung des Graphen erfolgt durch den Knoten $v_0 = (1, 0)$ einer Wurzel. Nun werden sukzessive unter Beachtung nachstehender Vorschriften den Relationen entsprechende neue Kanten bzw. Knoten eingefügt. Ich sage dazu auch "die Relation wird eingebaut".
Entsprechen (lp_1, lcp_1) und (lp_2, lcp_2) den Large Primes einer Relation, so werden auf dem Graphen folgende Operationen ausgeführt.

Schritt 1: Gibt es noch keinen Knoten $(lp_1, lcp_1) \in V$, so füge (lp_1, lcp_1) zu V hinzu. Analog verfahre mit (lp_2, lcp_2)

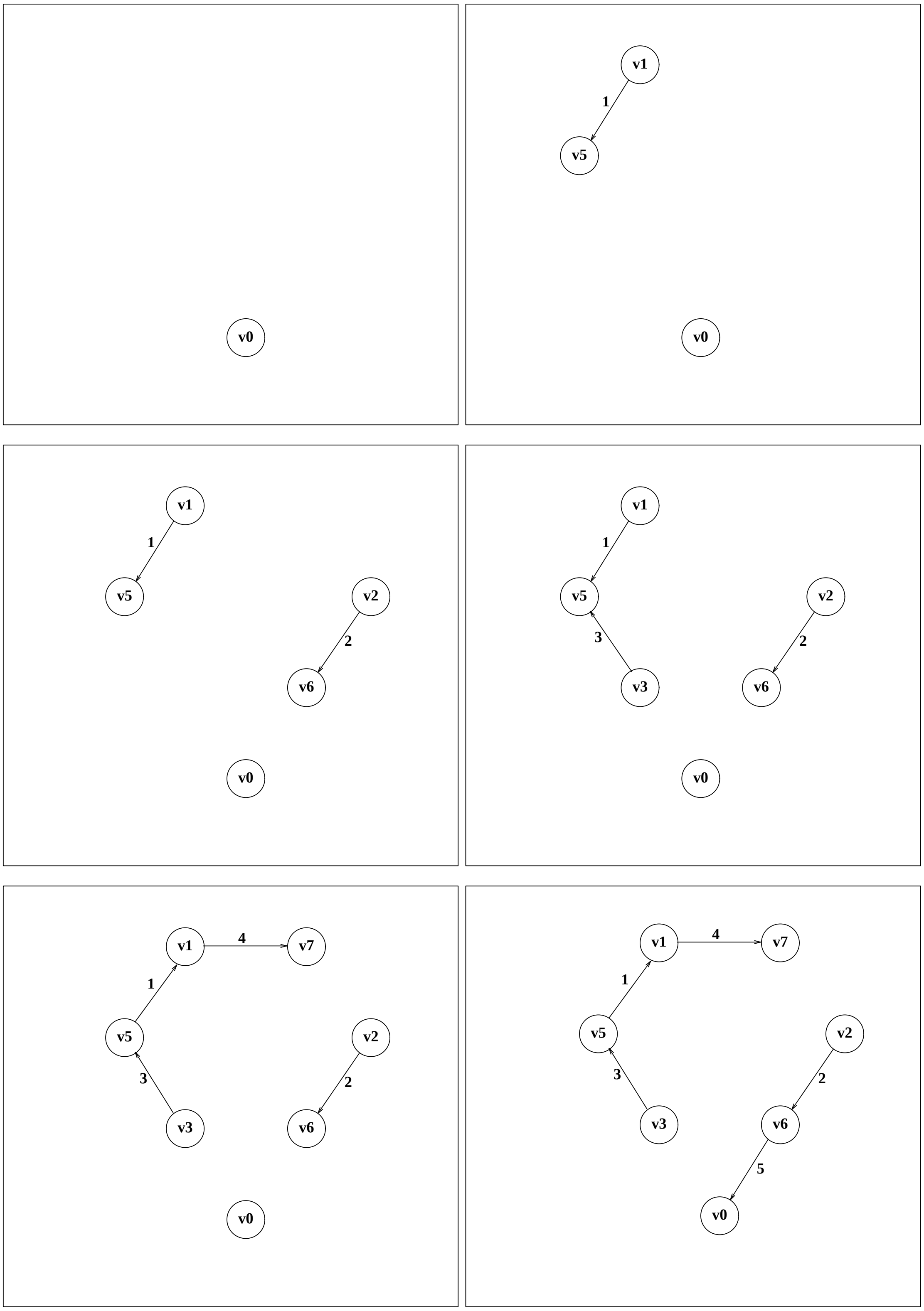
Schritt 2: Durchlaufe den Graphen von den Knoten $v_1 = (lp_1, lcp_1)$ und $v_2 = (lp_2, lcp_2)$ aus bis zu den entsprechenden Wurzeln w_1 und w_2 der jeweiligen Zusammenhangskomponente. O.B.d.A sei im weiteren $w_1 \leq w_2$.

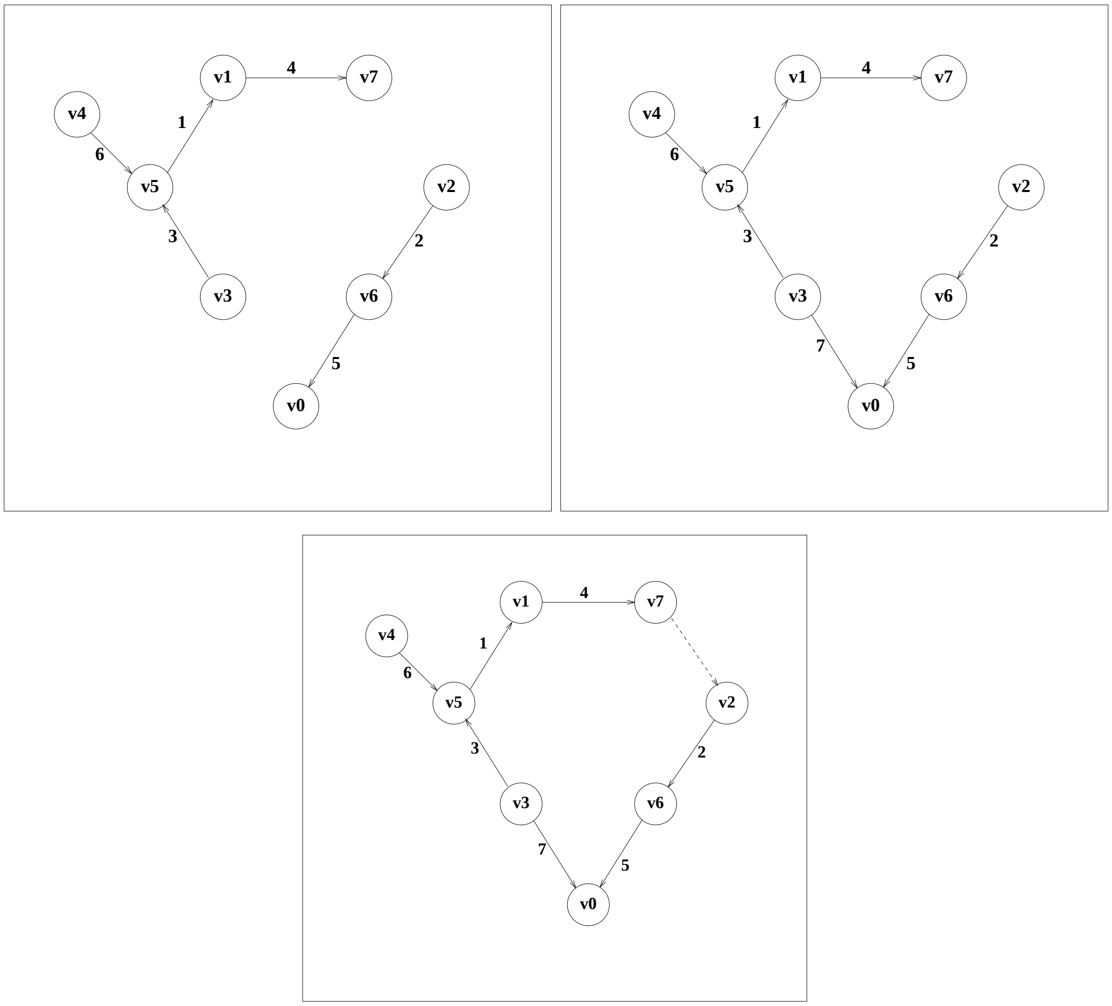
Ist $w_1 = w_2$, d.h., die beiden Knoten liegen in der gleichen Zusammenhangskomponente, so würde durch Einfügen der Kante $e = (v_1, v_2)$ in den Graphen ein Zykel geschlossen, wenn man die Richtungen aller Kanten unberücksichtigt läßt. Alle an diesem Zykel beteiligten Relationen werden herausgeschrieben, aber die Kante e wird, um Abhängigkeiten zu vermeiden, nicht in den Graphen aufgenommen.

Ist $w_1 < w_2$, so liegen die beiden Knoten in zwei verschiedenen Zusammenhangskomponenten. Diese werden nun durch Einfügen der Kante (v_2, v_1) miteinander verschmolzen. Die Wurzel dieser neuen Zusammenhangskomponente wird w_1 sein, die kleinere der beiden Wurzeln. Die Bedingungen, daß man von jedem Knoten einer Zusammenhangskomponente über die Kanten des Graphen zur Wurzel der Komponente gelangen kann, bzw. daß aus jedem Knoten höchstens eine Kante ausgeht, sind möglicherweise verletzt. Sie sind nur noch dann erfüllt, wenn w_2 gleich v_2 ist. Um diesen Fehler zu beheben, müssen alle Kanten auf dem Weg von v_2 zur Wurzel w_2 umgedreht werden.

Beispiel 3.11 (Aufbau des Graphen nach Beispiel 3.4)

Abbildung 3.3: Aufbau eines Graphen





Die 9 Abbildungen 3.3 zeigen sukzessive den Aufbau des Graphen aus Beispiel 3.4. Die Kanten sind dabei mit der Nummer der entsprechenden Relation gekennzeichnet. □

Bemerkung 3.12 (Aufbau des Graphen)
In der Praxis werden zwei Versionen dieses Verfahrens benutzt. Die erste wird nur zum Zählen der Zykel eingesetzt. Dies ist notwendig, da man während der Siebphase des NFS anhand der Relationen nicht zu erkennen vermag, ob schon genügend Zykel gefunden wurden. Die zweite Version wird schließlich dazu verwendet, um die Zykel wirklich zu berechnen. Der reine Zählalgorithmus ist an zwei Stellen wesentlich effizienter als die andere Version. Zum einen müssen die Zykel nicht mehr herausgeschrieben werden, also die entsprechenden Relationen nicht lokalisiert und die zugehörigen Exponentenvektoren nicht modulo 2 addiert werden, zum anderen kann man die Wege, die man von den Knoten zu ihren Wurzeln zurücklegen muß, abkürzen. Beim Zählen der Zykel ist es nämlich nicht von Interesse, welchen Weg man zur Wurzel nimmt, sondern nur, welches die Wurzel ist. Deshalb kann man alle Kanten auf dem Weg von einem Knoten zu seiner Wurzel durch Kanten, die direkt zur Wurzel gehen, ersetzen.
Im Schritt 2 heißt es demnach:
Ist $w_1 < w_2$, so drehe alle Kanten auf dem Weg von v_2 zur Wurzel w_2 nach w_1 um.

Ein wichtiger Faktor, der die Laufzeit des obigen Algorithmus betrifft, ist, wie schnell man feststellen kann, ob zu einer bestimmten Large Prime schon ein Knoten existiert und gegebenenfalls, welcher Knoten es ist.
In meiner Implementierung verwende ich dazu eine Hashtafel der Größe $H >$ Anzahl der verschiedenen Large Primes. Der Schlüssel, nach dem die Werte abgespeichert werden, sind die den Large Primes entsprechenden Paare (lp, lcp). Als einfache Hashfunktion bietet sich

$$\begin{array}{rrcl} h: & \mathbb{Z}^2 & \rightarrow & \{0,\ldots,H-1\} \\ & (lp,lcp) & \mapsto & (lp+lcp)\,Mod\,H \end{array} \tag{3.8}$$

an. Die Hashtafel besteht aus zwei eindimensionalen Feldern vom ganzzahligen Datentyp *long* (4 Byte) und dient zur Speicherung der Paare (lp, lcp). Kollisionen, also Situationen, in denen zwei Large Primes derselbe Hashwert und damit dieselbe Stelle in den Feldern zugewiesen wird, werden dadurch aufgelöst, daß die neu einzufügende Large Prime an die nächste freie Stelle der Hashtafel gesetzt wird.
Da aus jedem Knoten genau eine Kante ausgeht, genügt zum Speichern der Kanten ebenfalls ein Feld von *longs* der Länge H. Dies heißt also, daß insgesamt zum Speichern des Graphen

$$3 \cdot H \text{ longs} \tag{3.9}$$

also $12 \cdot H$ Bytes benötigt werden. Damit das Hashen aber effizient wird, sollte man eine Hashtafel verwenden, die wesentlich größer ist, z.B. $24 \cdot H$.

3.4 Die Quadruple Large Prime Variante

3.4.1 Das Prinzip

Die Idee der Verwendung von zwei Large Primes bei Faktorisierungsverfahren wurde zuerst im Quadratischen Sieb (QS) benutzt. Der im letzten Kapitel beschriebene Algorithmus wurde von A. K. Lenstra und M. Manasse auch ursprünglich für das QS entwickelt. Dort liegt aber eine andere Situation vor als im NFS Algorithmus, denn im QS gibt es nur eine Faktorbasis und somit auch nur eine Zerlegung, in der Large Primes vorkommen können. Allerdings ist man dort dazu übergegangen, nicht nur eine, sondern zwei Large Primes pro Zerlegung zu gestatten (siehe z.B. [De93]). Die Übertragung dieser Idee auf das *Number Field Sieve* , also das Erlauben von entweder zwei rationalen oder zwei algebraischen Large Primes pro Relation, wurde schon in [LLMP90, Kap. 7.3] diskutiert. Dort kam man aber zum Schluß, daß aufgrund der Vielzahl der durchzuführenden Probedivisionen diese Variante nicht praktikabel sei.
In den folgenden Abschnitten werde ich aufzeigen, welche Änderungen man im NFS Algorithmus durchführen muß, um sogar mit zwei Large Primes pro Zerlegung, also insgesamt vier pro Relation, effizient zu faktorisieren. Diese Variante des ursprünglichen Siebalgorithmus nenne ich *Quadruple Large Prime Variante* (QLP).
Läßt man zwei Large Primes pro Zerlegung zu, so macht dies eine Neudefinition des Begriffs *Relation* notwendig.

Definition 3.13 (gute Paare, Relation (Version 3))
Ein Paar $(a, b) \in \mathbb{Z}^2$ heißt gutes Paar *oder* ***Relation****, wenn für gegebene Faktorbasen FB_R und FB_A sowie Schranken LPBOUND, LPIBOUND $\in \mathbb{R}$ folgende drei Bedingungen erfüllt sind:*

$$\begin{aligned} \mathrm{ggT}(a, b) &= 1 \qquad (3.10) \\ a + bm &= \prod_{p \in FB_R} p^{e_p} \cdot lp_1 \cdot lp_2, \qquad (3.11) \end{aligned}$$

wobei für $i \in \{1, 2\}$ $lp_i = 1$ oder lp_i Primzahl kleiner LPBOUND ist, die nicht in der rationalen Faktorbasis FB_R liegt.

$$|N(a, b)| = \prod_{(p,cp) \in FB_A} p^{f(p,cp)} \cdot l\pi_1 \cdot l\pi_2, \qquad (3.12)$$

wobei für $i \in \{1, 2\}$ $l\pi_i = 1$ oder $l\pi_i$ eine Primzahl kleiner LPIBOUND ist, die nicht in einem Paar der algebraischen Faktorbasis vorkommt. □

Bemerkung 3.14 (Schreibweise)
Je nach Anzahl der Large Primes ungleich 1 und $< 1 >$ heißt ein gutes Paar (a, b)

Full Relation oder kurz **Full**,

Single Large Prime Relation oder kurz **Single**,

Double Large Prime Relation oder kurz **Double**,

Triple Large Prime Relation oder kurz **Triple** bzw.

Quadruple Large Prime Relation oder kurz **Quadruple**.

□

Die Klassifizierung der Relationen nach der Anzahl ihrer Large Primes läßt sich dadurch verfeinern, daß man noch unterscheidet, in welcher Anzahl rationale bzw. algebraische Large Primes vorkommen. Dazu verwende ich einen 4–stelligen Code, der nur die Buchstaben **f** (keine Large Prime) und **p** (Large Prime) enthält. Die ersten beiden Zeichen des Strings entsprechen der rationalen Seite, die letzten beiden Zeichen der algebraischen Seite. Somit entspricht z.B. ffff einer Full Relation, pppp einer Quadruple Large Prime Relation, fppp einer Triple Large Prime Relation mit einer rationalen und zwei algebraischen Large Primes.
Die Verwendung von zwei Large Primes pro Relation wirft mehrere Probleme auf:

Problem 1: Wie kann man anhand des Restes im Siebarray erkennen, ob eine Large Prime oder das Produkt zweier Large Primes vorliegt? Und wenn zwei Large Primes vorliegen, wie kann man die einzelnen Large Primes aus dem Produkt berechnen?

Problem 2: Wie bestimmt man Relationen mit Large Primes, deren Kombination ausschließlich zu Large Prime Quadraten führen?

Die Lösung dieser Probleme ist das Thema der nächsten Abschnitte. Dabei werde ich bei dem ersten Problem nur den rationalen Fall untersuchen. Der algebraische ergibt sich analog.

3.4.2 Erkennen der Large Primes

Sei $MAXP$ das größte Element der rationalen Faktorbasis und $LPBOUND$ eine obere Schranke für die zugelassenen Large Primes. Sei weiter r ein Rest, der nach Anwendung des RATSIEB–Algorithmus in einer Zelle des Siebarrays übrig geblieben ist. An der Größe von r kann man sehr oft schon erkennen, ob es sich um eine, um zwei oder um überhaupt keine Large Prime handelt. Dazu setze ich wiederum voraus, daß $MAXP^2 > LPBOUND$ ist. Dann gibt es vier interessante Werte.

1. $P_1 = MAXP$
2. $P_2 = LPBOUND$
3. $P_3 = MAXP^2$
4. $P_4 = LPBOUND^2$.

Diese Zahlen sind deshalb von Interesse, weil man je nach der Lage von r zu ihnen leicht ablesen kann, wie der Rest geartet ist:
Ist $r = 1$, so handelt es sich natürlich um keine Large Prime.
Ist $1 < r < MAXP$, so muß r nach Konstruktion in der Faktorbasis liegen. Dies ist aber nicht möglich, da im RATSIEB–Algorithmus mit allen Faktorbasiselementen gesiebt wurde. Dieser Fall kann also nicht vorkommen.

Liegt r zwischen $MAXP$ und $LPBOUND$, so kann es sich hier analog zu (3.3) nur um eine Primzahl handeln. r ist also eine Large Prime.

Für r zwischen $LPBOUND$ und $MAXP^2$ gilt das gleiche Argument: r ist eine Primzahl. Allerdings ist $r > LPBOUND$, also eine Primzahl, die nicht mehr als Large Prime akzeptiert wird.

Gilt $MAXP^2 < r \leq LPBOUND^2$, so gibt es zwei Möglichkeiten: entweder ist r eine Primzahl, die aufgrund ihrer Größe nicht als Large Prime anerkannt werden kann, oder r ist zusammengesetzt. Gilt nun, was in der Praxis immer zu erreichen ist, daß

$$MAXP^3 > LPBOUND^2$$

ist, so setzt sich r aus höchstens zwei Primzahlen zusammen. Zuerst muß also für

$$MAXP^2 < r \leq LPBOUND^2$$

überprüft werden, ob r eine Primzahl ist. Dazu kann man z.B. den Miller–Rabin Primzahltest verwenden, den ich auf Seite 167 näher erläutern werde. Erkennt dieser Test r als Primzahl an, so wird das entsprechende Paar (a, b) nicht weiter betrachtet, da r dann als Large Prime zu groß wäre. Die wenigen Fälle, in denen der Primzahltest versagt und eine zusammengesetzte Zahl als Primzahl bezeichnet (siehe Kapitel A.4), führen zu geringen Verlusten an Relationen, da die entsprechenden Relationen entfernt werden, obwohl sie möglicherweise Relationen mit zwei Large Primes auf der rationalen Seite sind. Ergibt der Primzahltest, daß es sich bei r um eine zusammengesetzte Zahl handelt, so müssen nun deren Primteiler bestimmt werden. Dazu werden Faktorisierungsalgorithmen verwendet, die besonders effizient für Zahlen dieser Größenordnung arbeiten. Hierzu kommen zum einen die Pollard ρ–Methode (siehe Anhang A.5) und zum anderen Shanks SQUFOF ([Ri85, Seite 191]) in Frage. Sind beide von diesen Verfahren berechneten Teilern kleiner als $LPBOUND$, so haben wir zwei Large Primes gefunden, ist aber ein Teiler größer als die Large Prime Schranke, so wird auf das entsprechende Paar (a, b) verzichtet.

Ist $r > LPBOUND^2$, so ist es gleich, ob r eine Primzahl oder zusammengesetzt ist. In jedem der beiden Fälle kommt wegen $\sqrt{r} > LPBOUND$ zumindest eine Primzahl vor, die größer als $LPBOUND$ ist. Das entsprechende Paar (a, b) bleibt also auf jeden Fall unberücksichtigt.

Um weiterhin jede Large Prime mit einem Paar (lp, lcp) identifizieren zu können, müssen wir noch die zweite Komponente des Paares berechnen. Diese ergibt sich aus der Bedingung

$$a + b \cdot lcp \equiv 0 \pmod{lp} \tag{3.13}$$

zu

$$lcp \equiv -a \cdot b^{-1} \pmod{lp}.$$

Die Bestimmung des Inversen von b ist natürlich nur möglich, wenn $\mathrm{ggT}(lp,\, b) = 1$ ist. Gilt dagegen $\mathrm{ggT}(lp,\, b) = lp$, so folgt aber direkt aus der Kongruenz (3.13), daß $lp \mid a$, also $\mathrm{ggT}(a,\, b) \neq 1$, was durch die Bedingung (3.10) verhindert wird.

Im Falle einer algebraischen Large Prime ist es dennoch möglich, daß $\mathrm{ggT}(lp,\, b) = lp$ ist, dann teilt lp nämlich den höchsten Polynomkoeffizienten f_D. Das gesuchte Large Prime Paar wäre dann (lp, lp).

Handelt es sich bei der gerade untersuchten Large Prime um eine rationale, so muß die zweite Komponente noch negiert werden, um keine Kollisionen zwischen rationalen und algebraischen Large Primes zu erhalten. Zusammengefaßt ergeben obige Betrachtungen folgenden Algorithmus.

Algorithmus 3.15 (Restchecker)

LPCHECK

EINGABE: $rest$, p_{max}, $lpbound$

AUSGABE:

$TRUE$, (l_1, lc_1), (l_2, lc_2),	wenn $l_1 \cdot l_2 = rest$
	und $p_{max} < l_1,\, l_2 \leq lpbound$
	oder $l_1 = l_2 = 1$
$FALSE$	sonst

```
if (rest = 1) then                                  /* Rest ist 1, also zerlegt */
  return (TRUE, (1, 0), (1, 0))
if (rest ≤ p_max) then                              /* Rest liegt in Faktorbasis */
  return (FALSE)
if (p_max < rest ≤ lpbound) then                    /* Rest ist Large Prime */
  if (rest | b) then                                /* spezielle Large Prime ? */
    return (TRUE, (rest, rest), (1, 0))
  else
    lc_1 := -a/b   (Mod rest)                       /* cp der Large Prime */
  return (TRUE, (rest, lc_1), (1, 0))
fi
if (lpbound < rest ≤ p_max^2) then                  /* Rest zu große Primzahl */
  return (FALSE)
if (p_max^2 < rest ≤ lpbound^2) then
  if (Miller–Rabin(rest) = TRUE) then               /* Rest zu große Primzahl */
    return (FALSE)
  else
    (l_1, l_2) := POLLARD_RHO(rest)                 /* faktorisiere Rest */
    if (l_1 | b) then                               /* spezielle Large Prime ? */
      lc_1 := l_1                                   /* cp der Large Prime l_1 */
    else
      lc_1 := -a/b   (Mod l_1)                      /* cp der Large Prime l_1 */
    if (l_2 | b) then                               /* spezielle Large Prime ? */
      lc_2 := l_2                                   /* cp der Large Prime l_2 */
    else
      lc_2 := -a/b   (Mod l_2)                      /* cp der Large Prime l_2 */
    return (TRUE, (l_1, lc_1), (l_2, lc_2))
  fi
else
  return (FALSE)                                    /* Rest zu groß */
fi
```

3.4.3 Kombinieren von Relationen mit maximal vier Large Primes

Entfernen unnützer Relationen

Das Kombinieren von Relationen mit mehr als zwei Large Primes wirft viele neue Probleme auf. Das erste besteht in der riesigen Menge der zu verwaltenden Daten. So wurden z.B. beim Sieben für die Zahl Z_{80} 5.61 Millionen Relationen mit insgesamt $\#LP = 15.48$ Millionen verschiedenen Large Primes generiert. Die Verwendung eines ähnlichen Verfahrens, wie es in Kapitel 3.3.2 vorgestellt wurde, würde mindestens (siehe (3.9))

$$12 \cdot 15.48 \cdot 10^6 \text{ Bytes} = 177.15 \text{ Megabyte} \tag{3.14}$$

Hauptspeicher erforden, der in der Praxis in den seltensten Fällen zur Verfügung steht. Allerdings gibt es unter den gefundenen Relationen sehr viele, die überhaupt keine Chance haben, daß ihre Large Primes zu Quadraten kombiniert werden. Dies sind Relationen mit Large Primes, die in keiner anderen Relation vorkommen. Deshalb ist es sinnvoll, dem eigentlichen Zykelbilden einen Reduktionsschritt voranzustellen, in dem solche Relationen entfernt werden.
Zum Erkennen solcher unnützer Relationen läßt sich wiederum eine Hashtafel mit der gleichen Hashfunktion wie in (3.8) verwenden. Allerdings wird in der Hashtafel lediglich lp von (lp, lcp) abgespeichert. Kollisionen werden ebenfalls wie oben aufgelöst. In seltenen Fällen können dadurch, daß in der Hashtafel die zweite Komponente lcp nicht mehr abgespeichert wird, verschiedene Large Primes auf dieselbe Stelle in der Hashrafel abgebildet werden. Dies bedeutet aber nur, daß Relationen, die möglicherweise hätten entfernt werden können, noch weiter betrachtet werden.
Zum Zählen der Häufigkeit der Vorkommen der verschiedenen Large Primes in den Relationen wird noch ein Feld vom Typ *unsigned char* (1 Byte) benötigt, das die gleiche Länge wie die Hashtafel aufweist.
Das Verfahren läuft in zwei Schritten. Im ersten Schritt werden alle vorkommenden Large Primes in die Hashtafel eingetragen und gezählt. Im zweiten Schritt werden alle Relationen gestrichen, in denen Large Primes vorkommen, deren Zähler auf 1 steht. Die Zähler aller Large Primes, die in dieser Relation vorkommen, werden um 1 dekrementiert. Hieraus ergeben sich Seiteneffekte, da durch das Dekrementieren Zähler von Large Primes anderer Relationen auf 1 gesetzt werden können. Aus diesem Grund muß dieser zweite Schritt sooft wiederholt werden, bis keine Relation mehr gestrichen werden kann.

Algorithmus 3.16 (Relationen Reduktion)

RELREDUCE

EINGABE: Menge T mit t Relationen
AUSGABE: Menge T mit Relationen, deren Large Primes mindestens zweimal vorkommen

PHASE 1: INITIALISIERUNG

(1) Initialsiere Hashtafel der Länge $4 \cdot t$
(2) Initialsiere Zählerfeld der Länge $4 \cdot t$

Fortsetzung auf Seite 62

PHASE 2: EINLESEN DER RELATIONEN

```
for (alle Relationen r aus T) do
   for (jede Large Prime (p, cp) von r) do
      Berechne Position von (p, cp) in der Hashtafel
      Inkrementiere den entsprechenden Zähler
   od
od
```

PHASE 3: ENTFERNEN UNNÜTZER RELATIONEN

```
repeat
   for (alle Relationen r aus T) do
      for (jede Large Prime (p, cp) von r) do
         Berechne Position von (p, cp) in der Hashtafel
      od
      if ( Der Zähler mindestens einer Large Prime von r ist 1 ) then
         dekrementiere den Zähler aller Large Primes von r um 1
         streiche r aus T
      fi
   od
until (keine Relation wurde aus T getrichen)
return (T)
```

Nach diesem Reduktionsschritt bleiben im Beispiel Z_{80} nur noch 1.93 Millionen Relationen mit 1.49 Millionen verschiedenen Large Primes übrig. Der Aufbau eines Graphen zu diesen Relationen erfordert nur noch 17.05 Megabyte Hauptspeicher. Der Reduktionsalgorithmus selbst benötigt für Hashtafel und Zähler selbst mindestens $\#LP \cdot (4+1)$ Bytes, was in diesem Falle etwa 73.81 Megabyte entspricht. Je größer man die Hashtafel wählt, umso weniger Kollisionen gibt es und umso schneller kann für eine Large Prime ihr Platz in der Hashtafel bestimmt werden. Ist man nicht bereit bzw. ist es unmöglich, soviel Hauptspeicher zu investieren, so läßt sich ein Vorreduktionsschritt durchführen, der mit wesentlich weniger Hauptspeicher auskommt, aber auch nur Teile der unnützen Relationen entfernt. Dieses Verfahren verwendet ebenfalls eine Hashtafel mit obiger Hashfunktion (3.8). Der Unterschied besteht darin, daß im Vorreduktionsschritt die Hashtafel die Funktion des Zählers übernimmt und keine Kollisionsauflösung durchgeführt wird. Die Large Primes werden mittels Hashfunktion gehasht und der entsprechende Eintrag in der Hashtafel hochgezählt. Nachdem auf diese Weise alle Relationen bearbeitet wurden, werden analog zu obigem Reduktionsalgorithmus, alle Relationen mit Large Primes, deren Zähler 1 ist, gelöscht. Da keine Kollisionsauflösung betrieben wird, besteht die Möglichkeit, daß verschiedene Large Primes den gleichen Hashwert und somit den gleichen Zähler haben. Dadurch können Relationen diesen Reduktionsschritt über-

stehen, obwohl sie Large Primes beinhalten, die sonst nirgends vorkommen. Diese werden aber im Hauptreduktionsalgorithmus herausgestrichen.
Da in diesem Vorreduktionsschritt die Hashtafel nur aus einem Zähler besteht, muß auch nur pro Large Prime ein Byte spendiert werden. Im Z_{80}–Beispiel benötigt man also mindestens 14.76 Megabyte. Je größer die Hashtafel aber gewählt wird, um so geringer ist die Wahrscheinlichkeit, daß verschiedene Large Primes auf die gleiche Stelle gehasht werden. Dadurch erhöht sich die Anzahl der Relationen, die gestrichen werden, wie Tabelle 3.3 bestätigt.

Tabelle 3.3: Vorreduktionsschritt am Beispiel Z_{80}

Größe der Hashtafel	Anzahl der über–lebenden Relationen
$2 \cdot 10^7$	2849678
$3 \cdot 10^7$	2642221
$4 \cdot 10^7$	2349779
$5 \cdot 10^7$	2263805
$6 \cdot 10^7$	2233272
$7 \cdot 10^7$	2148180
$8 \cdot 10^7$	2111336

Der zweite Schritt des Vorreduktionsalgrithmus sollte aus Effizienzgründen in der Praxis nicht solange durchgeführt werden, bis keiner der Zählerwerte mehr auf 1 steht, denn die Anzahl der entfernten Relationen nimmt in den von mir untersuchten Beispielen pro Durchlauf um einen Faktor größer gleich 2.5 ab (siehe Tabelle 3.4).
Da in einem Schritt alle verbliebenen Relationen betrachtet werden müssen, ist es sinnvoll, die Vorreduktion abzubrechen und mit der Hauptreduktion zu beginnen, wenn pro Schritt beispielsweise weniger als 1000 Relationen entfernt werden. Zieht man die Vorreduktion bis zum Ende durch, so benötigt der Reduktionsalgorithmus noch mindestens 10.65 Megabyte zum Abspeichern der Hashtafel. Bei einem Abbruch nach sechs Reduktionsrunden würden dagegen 10.66 Megabyte benötigt, was keinen großen Unterschied macht.
Der Hauptreduktionsalgorithmus verringert die Anzahl der übriggebliebenen Relationen wiederum deutlich. Im Beispiel Z_{80} überleben nur noch 1929533 Relationen mit 1488563 verschieden Large Primes diesen Schritt. Dies entspricht im Vergleich zu den 177 Megabyte aus (3.14) einem Hauptspeicherbedarf von 16.94 Megabyte für das Abspeichern des Graphen im Zykelzählalgorithmus.
Im weiteren werde ich zwei Alternativen benennen, wie man die verbliebenen Relationen kombinieren muß, damit alle Large Primes in gerader Potenz vorkommen.
Eine einfache Möglichkeit ist der Aufbau und das Lösen eines linearen Gleichungssystems über dem Körper mit zwei Elementen. Dazu werden die auftretenden Large Primes bei 1 startend fortlaufend numeriert. Das System wird so erstellt, daß die

Tabelle 3.4: Abnahme der Relationen im Vorreduktionsschritt am Beispiel Z_{80} bei einer Hashtafel von 60 Millionen Einträgen

Durchlauf	Anzahl der über- lebenden Relationen	Anzahl der ent- fernten Relationen
-	5609255	-
1	3030362	2578893
2	2454194	576168
3	2299440	154754
4	2253081	46359
5	2239310	13771
6	2235152	4158
7	2233859	1293
8	2233455	404
9	2233319	136
10	2233278	41
11	2233272	6

j–te Spalte der j–ten Relation und die i–te Zeile der i–ten Large Prime entspricht. Der Eintrag an der Stelle (i, j) ist gleich der Häufigkeit, mit der die i–te Large Prime in der Relation j vorkommt.
Jede lineare Abhängigkeit in diesem Gleichungsystem führt zu einer gewünschten Kombination der Large Primes. Das Gleichungssystem ist zwar dünnbesetzt (Maximal vier Einträge pro Spalte), hat aber eine riesige Dimension. Im Beispiel der 80–stelligen Zahl müßte ein System mit 1488563 Zeilen und 1929533 Spalten gelöst werden, was mit den aktuellen Algorithmen nicht mehr in vertretbarer Zeit zu leisten ist.
Die zweite Variante, die ich nun vorstellen möchte, stellt eine Erweiterung des in Kapitel 3.3.2 beschriebenen Graphenalgorithmus dar. Ich werde aber zwischen dem Zählen und dem Berechnen der Zykel unterscheiden. In beiden Fällen wird allerdings wie oben ein Graph aufgebaut, dessen Knoten den Paaren (lp, lcp) entsprechen. Aus algorithmischen Gründen ist der Graph gerichtet, gesucht werden aber Zykel des gleichen Graphen ohne Berücksichtigung der Richtungen der Kanten.

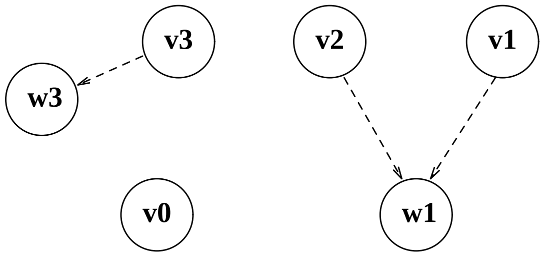
Zählen der Zykel

Das Zählen der Zykel für Relationen mit bis zu vier Large Primes erfolgt in mehreren Phasen. Zuerst wird der Graph mit der Wurzel $v_0 = (1, 0)$ initialisiert. Dann werden alle Large Primes von Single und Double Relationen analog zu obigem Verfahren eingetragen bzw. dadurch entstandene Zykel gezählt. Die Kante, die einen Zykel schließt, wird aber nicht in den Graph aufgenommen. In der zweiten Phase wird versucht, auch Triple und Quadruple Relationen einzubauen. Die Large Primes solcher Relationen werden aber nur dann in den Graph aufgenommen, wenn durch Bildung von Teilzykeln alle bis auf höchstens zwei zu Quadraten gemacht werden können.

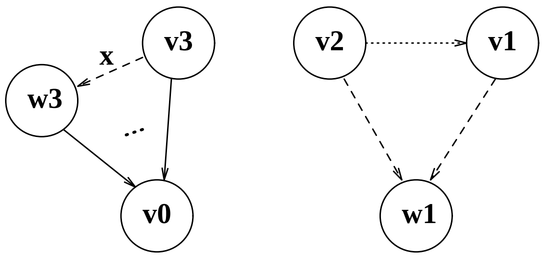
Bemerkung 3.17 (Sprechweise)
Je nach Anzahl der verbliebenen Large Primes sagen wir, die Triple (Quadruple) Relation konnte zu einer Double, Single bzw. Full Relation kombiniert werden. □

Beispiel 3.18 soll dies verdeutlichen. Bei den zugehörigen Abbildungen ist zu beachten, daß die gestrichelten Linien den schon im Graph vorhandenen Kanten bzw. durch den Graph führenden Wegen entsprechen. Die gepunkteten Linien stehen für Kanten, die einen Teilzykel schließen, während die durchgezogenen Linien die neuentstandenen Kanten darstellen. Die mit x beschrifteten Kanten bezeichnen die Kanten, die durch den Einbau der aktuellen Relation gelöscht werden.

Beispiel 3.18 (Einbau einer Triple Relation in den Graphen)
Es sei folgender Graph schon aufgebaut:



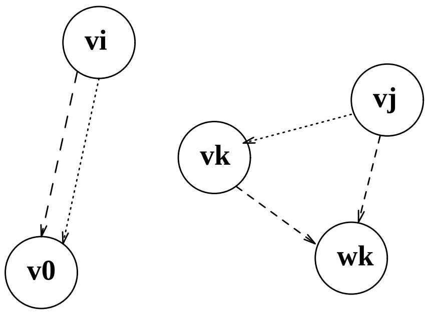
Eine Triple Relation, deren Large Primes den Knoten v_1, v_2, v_3 entsprechen, soll aufgenommen werden. Dies ist möglich, da die Knoten v_1 und v_2 die gleiche Wurzel w_1 besitzen und somit in der Zusammenhangskomponente mit der Wurzel w_1 einen Zykel schließen. Dadurch reduziert sich die Anzahl der Large Primes, die nicht in gerader Potenz auftreten, auf eins. Man kann diese Triple Relation also wie eine Single Relation mit einer dem Knoten v_3 entsprechenden Large Prime behandeln. Da die Wurzeln w_3 und v_0 nicht identisch sind, werden die beiden zugehörigen Zusammenhangskomponenten verschmolzen. Gemäß der Bemerkung 3.12 hat dies zur Folge, daß alle Kanten, die auf dem Weg vom Knoten v_3 zu seiner Wurzel w_3 liegen, nach v_0 umgebogen werden:



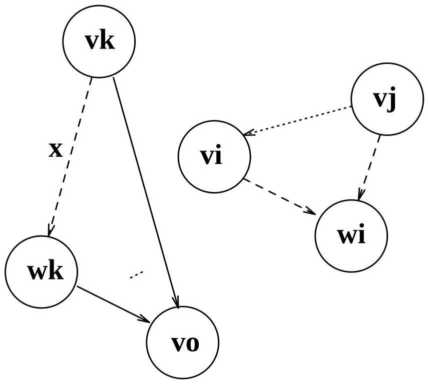
□

Zum Testen, ob die Large Primes einer Triple Relation in den Graph aufgenommen werden, genügt es, die Wurzeln der ihnen entsprechenden Knoten zu untersuchen. Seien v_i, v_j und v_k die einer Triple Relation entsprechenden Knoten. Jeder Knoten, der noch nicht in den Graph aufgenommen wurde, wird nun eingetragen. Er bildet zunächst eine eigene Zusammenhangskomponente. Seien weiter w_i, w_j und w_k die Wurzeln der drei Knoten, so gibt es folgende Möglichkeiten

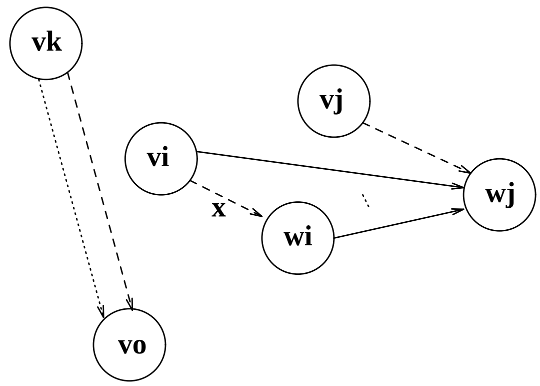
1. Sind zwei der Wurzeln gleich und die dritte v_0, so kann die Relation zu einer Full kombiniert werden, da alle Large Primes durch die beiden Zykel kombiniert werden können. Dies bedeutet, daß die Relation nicht in den Graph aufgenommen wird, die Anzahl der Zykel sich aber um 1 erhöht. Im Beispiel ist $w_i = v_0$ und $w_j = w_k$.



2. Sind zwei der Wurzeln gleich, die dritte aber ungleich v_0, so kann die Relation zu einer Single kombiniert werden, da zwei der drei Large Primes in einer Zusammenhangskomponente liegen und somit ein Teilzykel geschlossen werden kann. Die Kanten von der dritten Large Prime zu ihrer Wurzel werden nach v_0 gedreht. Im Beispiel ist $w_i = w_j$ und $w_k \neq v_0$



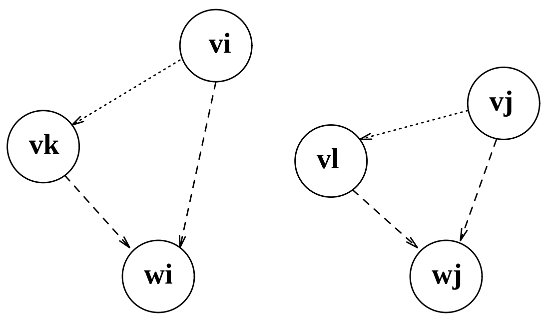
3. Sind die drei Wurzeln paarweise ungleich, eine aber gleich v_0, so kann die Relation zu einer Double kombiniert werden, da die Large Prime mit der Wurzel v_0 zu einem Quadrat kombiniert werden kann. Die Zusammenhangskomponenten, in denen die beiden verbliebenen Knoten liegen, werden miteinander verschmolzen. Dazu werden alle Kanten, die auf dem Weg von dem Knoten mit der größeren Wurzel zu dieser Wurzel liegen, zur kleineren Wurzel gedreht. Des Weiteren wird eine Kante von diesem Knoten zur kleineren Wurzel eingefügt. Im Beispiel ist $w_i > w_j$ und $w_k = v_0$.



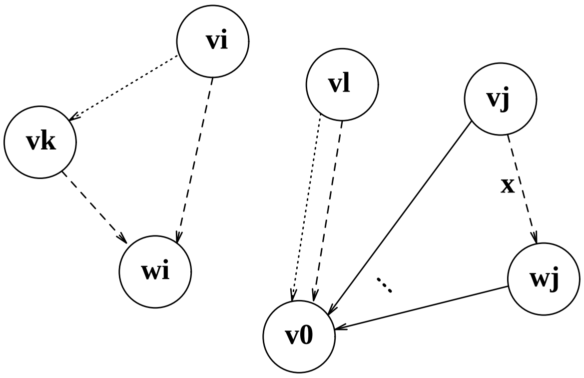
4. Sind die drei Wurzeln paarweise ungleich und keine ist gleich v_0, d.h. die drei Knoten liegen in drei verschiedenen Zusammenhangskomponenten, so kann die Relation nicht in den Graph eingebaut werden. Sie wird gespeichert und man versucht zu einem späteren Zeitpunkt wieder, sie in den Graph aufzunehmen.

Der Einbau von Quadruple Relationen erfolgt analog. Seinen v_i, v_j, v_k und v_l die Knoten und w_i, w_j, w_k, w_l die entsprechenden Wurzeln, so lassen sich die Relationen gemäß ihren Wurzeln wie folgt eintragen:

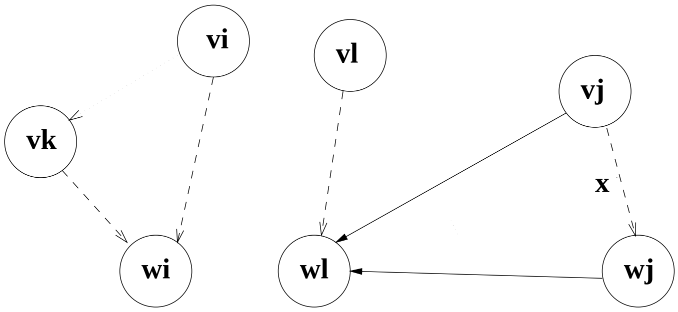
1. Wenn je zwei der vier Wurzeln gleich sind, so kann die Quadruple Relation zu einer Full kombiniert werden, da je zwei der Knoten in derselben Zusammenhangskomponente liegen, und somit alle Large Primes durch zwei Teilzykel kombiniert werden können. Im Beispiel ist $w_i = w_k$ und $w_j = w_l$.



2. Wenn zwei Wurzeln gleich sind, die anderen beiden ungleich, eine davon aber gleich v_0, so können drei der vier Large Primes durch zwei Teilzykel wegkombiniert werden. Es bleibt also eine Single Relation übrig, die in den Graph aufgenommen wird. Dabei werden alle Kanten auf dem Weg vom Knoten dieser Large Prime zu ihrer Wurzel nach v_0 gedreht. Im Beispiel ist $w_i = w_k$, $w_l = v_0$ und $w_j > v_0$.



3. Wenn zwei Wurzeln gleich sind, die anderen beiden ungleich und zusätzlich ungleich v_0, so können zwei der Large Primes durch einen Teilzykel wegkombiniert werden. Es verbleibt also eine Double Relation, die zwei Zusammenhangskomponenten miteinander verbindet. Deshalb werden alle Kanten, die auf dem Weg von dem Knoten mit der größeren Wurzel zu dieser Wurzel liegen, zur kleineren Wurzel gedreht. Des Weiteren wird eine Kante von diesem Knoten zur kleineren Wurzel eingefügt. Im Beispiel ist $w_i = w_k$ und $w_j > w_l > v_0$.



4. Sind die Wurzeln paarweise ungleich, so kann die Relation nicht in den Graph eingebaut werden. Sie wird gespeichert und man versucht, sie zu einem späteren Zeitpunkt in den Graph aufzunehmen.

Der gerade beschriebenene Algorithmus findet nicht immer alle unabhängigen Zykel, z.B. bleiben solche, die nur aus Triple und Quadruple Relationen aufgebaut werden, unberücksichtigt, wenn sie unabhängig von anderen gefundenen Zykeln sind. Dies ist aber nur mit Relationen möglich, die nicht in den Graphen eingebaut werden können. Im Beispiel Z_{80} konnten aber alle Relationen, die den Reduktionsschritt passiert haben, in den Graphen eingebaut werden, was bedeutet, daß alle unabhängigen Zykel gefunden werden.
Können nicht alle Relationen in den Graph eingebaut werden, so zeigt die Anwendung des Reduktionsschrittes auf alle Relationen außer denen, die nicht in den Graph aufgenommen wurden, weil sie einen Zykel schlossen, daß nur noch wenige Relationen zum Bilden weiterer Zykel in Frage kommen. In allen von mir durchgeführten Faktorisierungen war es so, daß zu dem Zeitpunkt, als genügend viele Zykel gefunden worden waren, alle Relationen in den Graph aufgenommen werden konnten.

Bilden der Zykel

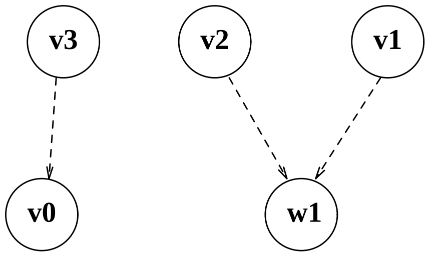
In diesem Abschnitt möchte ich aufzeigen, wie man mit Hilfe eines Graphenalgorithmus auch das Bilden der Zykel realisieren kann. Dabei wird es wieder so sein, daß ein gerichteter Graph aufgebaut wird, als Zykel aber solche gelten werden, die durch den gleichen Graph ohne Beachtung der Richtung der Kanten erzeugt werden.
Zuerst wird der Graph mit all den Wurzeln initialisiert, die nach der Ausführung des Zählalgorithmus noch in jenem Graph vorhanden sind. In fast allen von mir untersuchten Graphen, die aus Relationen von Faktorisierungen entstanden sind, blieb nur noch eine Wurzel übrig, nämlich $v_0 = (1, 0)$. Die Ausnahme bildeten Wurzeln, die sich als einziger Knoten in ihrer Zusammenhangskomponente befanden. Diese Wurzeln stammen von Relationen, in denen eine Large Prime schon als Quadrat vorkommt.
Nacheinander werden nun alle Relationen eingebaut. Dabei wird eine Relation nur dann in den Graph aufgenommen, wenn alle Knoten über Kanten an die bestehenden Zusammenhangskomponenten angebunden werden können. Betrachtet man, wie die Zusammenhangskomponenten im Zählalgorithmus entstanden sind, so sieht man, daß eine Relation im schlechtesten Fall in zwei verschiedenen Komponenten liegen kann. Bei Single und Double ist dies klar. Triple und Quadruple müssen, um aufgenommen zu werden, zumindest einen Teilzykel mit einer Large Prime (im Triple Fall) oder einen Teilzykel mit zwei Large Primes (im Quadruple Fall) liefern. Somit können höchstens noch zwei Large Primes übrig bleiben. Diese liegen aber nach Abschluß der Einfügeoperationen in einer Komponente, denn es gibt nur drei Möglichkeiten.

1. Verbleiben nur eine Large Prime oder zwei Large Primes, die schon in einer Komponente liegen, so ist die Behauptung direkt erfüllt.

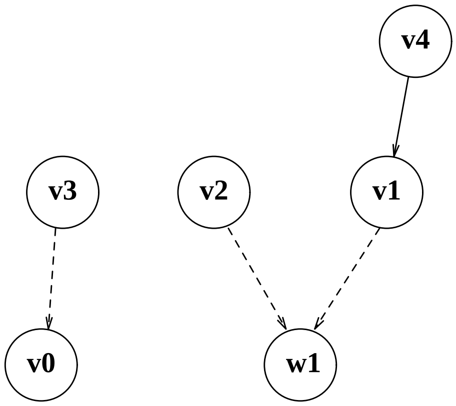
2. Der Knoten einer der beiden verbliebenen Large Primes liegt in der Komponente mit der Wurzel $(1, 0)$. Dann wird diese Large Prime zum Quadrat kombiniert, und es bleibt nur noch eine Large Prime übrig. Diese wird als Single eingebaut, liegt somit aber ebenfalls in der Komponente mit der Wurzel $(1, 0)$.

3. Liegen beide Large Primes in verschiedenen Komponenten und keine der beiden in der Komponente mit der Wurzel $(1, 0)$, so werden die beiden Knoten miteinander verbunden und liegen somit in der gleichen Komponente.

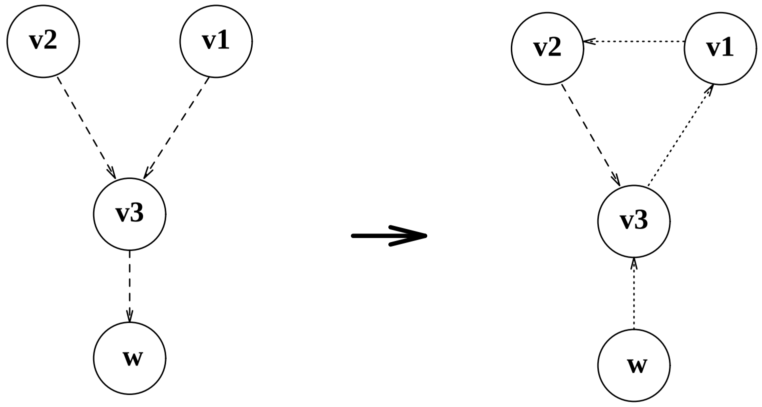
Soll eine Kante in den Graph eingebaut werden, ohne daß auch ein neuer Knoten integriert wird, so würde diese Kante einen (Teil–)Zykel schließen. Dieser Zykel wird berechnet, die schließende Kante aber nicht in den Graphen eingebaut. Damit ist der Graph jederzeit azyklisch.
Die Kanten neu eingefügter Knoten werden so gerichtet, daß sie vom neuen Knoten aus in den bestehenden Teilgraph gehen.
So kann zum Beispiel in den Graph



die Double Relation mit den Knoten v_1 und v_4 aufgenommen werden,



da der neue Knoten v_4 an den Teilgraph mit der Wurzel w_1 angebunden werden kann. Dagegen läßt sich die Double Relation mit den Knoten v_5 und v_6 zu diesem Zeitpunkt noch nicht einfügen.
Relationen, die nicht in den Graph aufgenommen werden können, müssen gespeichert und zu einem späteren Zeitpunkt eingebaut werden. Da alle Wurzeln des Zählgraphen im aktuellen Graphen vorhanden sind, lassen sich auch alle Relationen, die der Zählgraph beinhaltet, in den aktuellen Graph einbauen.
Soll eine neue Kante, aber kein neuer Knoten eingebaut werden, d.h. ist ein Zykel gefunden, so müssen noch die entsprechenden Relationen, die zum Zykel führen, in Erfahrung gebracht werden. Dazu ist es zuerst einmal notwendig, die Kanten zu berechnen, die an dem Zykel beteiligt sind.
Sei (v_1, v_2) eine Kante, die einen Zykel schließt. Dann lassen sich die Knoten und Kanten des Zykels sehr leicht dadurch berechnen, daß man alle Kanten von v_1 zur Wurzel des Teilgraphen umdreht und die Kante (v_1, v_2) einfügt. Dadurch entsteht auch im gerichteten Graph ein Zykel. Die gedrehten Kanten sowie die, die einen Zykel schließen, sind in der weiteren Abbildungen durch gepunktete Linien dargestellt.

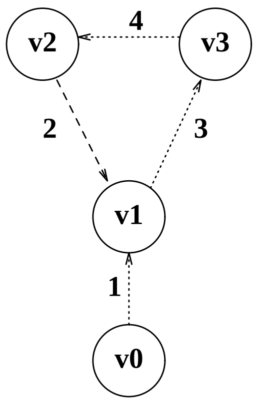


Nähme man nur Single und Double Relationen in den Graph auf, so würde eine einfache Beschriftung der Kanten mit der Nummer der Relation, die sie erzeugt hat, ausreichen, um die an der Kombination der Large Primes beteiligten Relationen abzulesen. Der Zykel muß dann nämlich nur noch von v_1 nach v_1 durchlaufen werden, wobei die zur Beschriftung der Kanten gehörenden Relationen miteinander zu verarbeiten sind.

Beispiel 3.19 (Zykelbilden mit Single und Double Relationen)
Es sei die Zuordnung von Knoten zu den Large Primes der Relationen wie folgt, wobei v_0 wiederum $(1, 0)$ entspricht.

1. v_1
2. v_1 v_2
3. v_1 v_3
4. v_2 v_3

Der zugehörige Graph mit Beschriftung der Kanten sieht nach dem Einbau der vier Relationen und dem dadurch entstandenen Zykel wie folgt aus.



Die Relationen 2 bis 4 liefern also eine gesuchte Kombination. □

Durch die Hinzunahme von Triple und Quadruple Relationen genügt die Beschriftung der Kanten mit der Nummer der Relation nicht mehr. Es müssen alle an den Teilzykeln der Triple oder Quadruple Relation beteiligten Relationen ebenfalls in die Beschriftung der Kante aufgenommen werden.

Beispiel 3.20 (Zykelbilden inklusive Triple und Quadruple Relationen)
Es sei die Zuordnung von Knoten zu den Large Primes der Relationen wie folgt, wobei v_0 wiederum $(1, 0)$ entspricht und w die Wurzel eines Teilgraphens ist.

1. v_1
2. v_1 v_2
3. v_1 v_3

4. $v_4\ w$

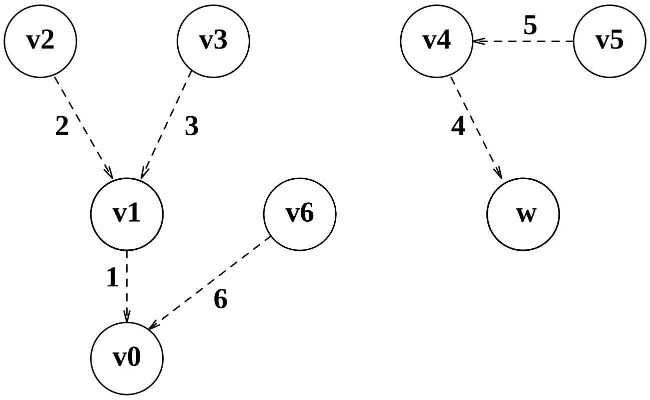
5. $v_4\ v_5$

6. v_6

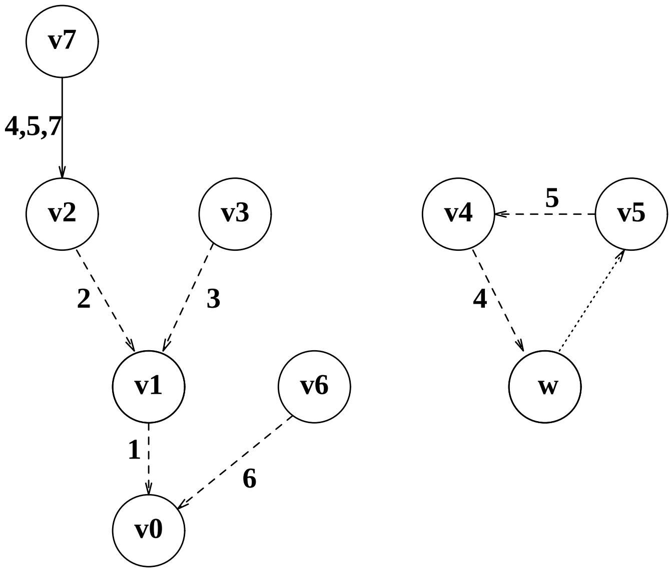
7. $v_2\ v_5\ v_7\ w$

8. $v_3\ v_6\ v_7$

Der zugehörige Graph vor dem Einbau der siebten Relation ist



Durch Relation 7 wird der Teilzyklus v_5, v_4, w in der Zusammenhangskomponente mit der Wurzel w geschlossen. Die Quadruple Relation wird zu einer Double gemacht. Die neu einzufügende Kante (v_2, v_7) wird mit den Nummern der Relationen des Teilzykels und der aktuellen Relation beschriftet.



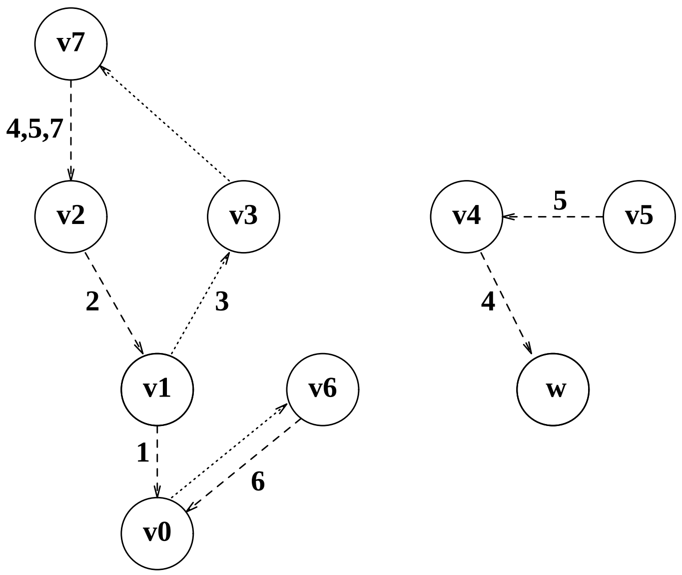
Die Triple Relation 8 schließt zwei Teilzykel:

$$v_7,\ v_2,\ v_1,\ v_3,\ v_7$$

bzw.

$$v_6,\ v_0,\ v_6.$$

Durch den ersten wird die Triple Relation zu einer Single mit der Kante (v_6, v_0). Diese wird mit 4, 5, 7, 2, 3, 8 beschriftet. Baut man diese Kante in den Graph ein, so entsteht ein zweiter Teilzykel. Insgesamt wurde also eine Full Relation kombiniert, die sich aus den letzten sieben Relationen zusammensetzt.



□

3.4.4 Effizienz der Quadruple Large Prime Variante

In diesem Kapitel möchte ich mich mit der Frage beschäftigen, ob die Verwendung von maximal vier Large Primes pro Relation das Faktorisieren beschleunigt.

Die Laufzeit der Siebphase

Einen großen Einfluß auf die Laufzeit nimmt diese Variante in erster Linie im Probedivisionsalgorithmus. Verwendet man den LOGSIEB–Algorithmus, so bleiben nach der Anwendung der Siebalgorithmen noch sehr viele Paare übrig, die möglicherweise ein gutes Paar sind. Um herauszufinden, welche Paare wirklich die Bedingungen von Definition 3.13 erfüllen, müssen die Siebalgorithmen so verändert werden, daß sie auch Paare mit je zwei rationalen und algebraischen Large Primes als gute Paare akzeptieren. Dies kann durch geschicktes Setzen der Schranke KORRFAK ermöglicht werden. Für die Laufzeit des Siebschrittes macht dies allerdings keinen Unterschied. Leider läßt sich aber durch dieses Vorgehen nur erreichen, daß die Reste, die in den Siebarrayeinträgen verbleiben, in einem gewissen Bereich liegen. Es wird dadurch aber keineswegs garantiert, daß diese Reste auch wirklich zu Relationen mit Large Primes gehören. Um dies herauszufinden, muß man einen modifizierten Probedivisionsalgorithmus (vgl. Algorithmus 3.1) auf diese Paare anwenden. Dieser muß den rationalen und den algebraischen Wert eines Paares (a, b), das die Siebalgorithmen als möglicherweise gut erkannt haben, über den Faktorbasen zerlegen. Je nachdem, wie groß die Reste sind, müssen sie einem Primzahltest unterzogen und gegebenenfalls faktorisiert werden. Der folgende Algorithmus leistet dies und bestimmt zusätzlich die ersten $B1 + B2 + 1$ Einträge des Exponentenvektors gemäß Definition 2.23 von Seite 33.

Algorithmus 3.21 (modifizierte Probedivision)

MODIFIZIERTE PROBEDIVISION

EINGABE: a, b, $h(x)$, FB, $lpbound$
AUSGABE:

$$\begin{array}{rl} \vec{e} = \left(e_1^{(a,b)}, \ldots, e_{|FB|+1}^{(a,b)}\right) \text{ und} & \\ \text{bis zu zwei Large Primes,} & \text{wenn } N_h(a,b) \text{ über } FB \text{ bis auf} \\ & \text{2 Large Primes zerfällt} \\ FALSE, & \text{sonst} \end{array}$$

PHASE 1: INITIALISIERUNG

$\vec{e} := (0, \ldots, 0)$ /* Exponentenvektor */
$lp_1 := lp_2 := 1$ /* Large Primes */

PHASE 2: ZERLEGUNG ÜBER DER FAKTORBASIS

$val := N_h(a,b)$ /* Initialisierung */
if $(val < 0)$ **then** /* negative Norm? */
  $val := -val$; $e_{|FB|+1}^{(a,b)} := 1$ /* ändere Vorzeichen */
foreach $((p_i, cp_i) \in FB)$ **do** /* für alle Faktorbasis-elemente (p_i, cp_i) */
  if $(a + b \cdot cp_i \equiv 0 \pmod{p_i})$ **then** /* Teilt (p_i, cp_i)? */
    repeat
      $e_i^{(a,b)}$ ++ /* Erhöhen des Exponenten */
      $val := val/p_i$ /* Herausdividieren der */
    until $(p_i \nmid val)$ /* maximalen Potenz von p_i */
  fi
od

PHASE 3: ÜBERPRÜFUNG DES RESTES

if (LPCHECK($val, p_{|FB|}, lpbound, lp_1, lp_2$) = FALSE) **then** /* $a + bm$ zerfällt nicht über der Faktorbasis */
  return (FALSE)
else
  return $(\vec{e}, lp_1, lp_2)$

Die Laufzeit dieses Algorithmus wird durch die Vielzahl der Divisionen dominiert, wie das folgende Beispiel zeigt.

Beispiel 3.22 (Laufzeit der Probedivision)
Messungen während der Siebphase für eine 89–stellige Zahl Z_{89} ergeben, daß für $b = 7500$ und die Schranken $A_{min} = -A_{max} = -10^7$ bei $|FB_R| = 25000$ und $|FB_A| = 55000$ die Relationensuche etwa 900 Sekunden dauert. 94% davon, etwa 846 Sekunden, entfallen alleine auf die Probedivision. Nach dem LOGSIEB–Algorithmus bleiben dabei 1599 Kandidaten übrig. Durch die modifizierte Probedivision mit der rationalen Faktorbasis reduziert sich diese Zahl auf 311 und nach dem Zerlegen des algebraischen Wertes auf 23. Dabei handelt es sich um 8 Quadruple, 10 Triple und 5 Double Relationen, von denen 3 vom Typ fpfp sind. Diese drei werden auch gefunden, wenn man nur mit einer Large Prime pro Zerlegung arbeitet. Diese Variante benötigt nur 50.52 Sekunden für das Sieben, wofür in erster Linie die geringe Anzahl von den Paaren verantwortlich ist, die nach dem Sieben mit Logarithmen noch akzeptiert werden. Diesen Part überleben nämlich nur 61. Davon verbleiben 3 nach der Probedivision mit der algebraischen Faktorbasis, und eben diese drei werden schließlich als gute Paare erkannt. Für die Probedivision mußten in diesem Fall nur 30.23 Sekunden aufgebracht werden. □

Im obigen Beispiel benötigt man bei beiden Varianten etwa 20 Sekunden für das eigentliche Sieben. Die fehlenden 34 Sekunden im Quadruple Fall werden für die Primzahltests und das Berechnen der Faktoren aus dem Produkt der zwei Large Primes benötigt. Diese Sekunden sind auf jeden Fall zu zahlen, wenn man mit vier statt zwei Large Primes arbeiten will. Berechnet man die Zeit, die benötigt wird, um eine Relation zu bestimmen, so ergibt sich für die Quadruple Large Prime Variante 39.13 Sekunden pro Relation und im anderen Fall 16.84 Sekunden pro Relation. Da außerdem die Relationen mit mehr als zwei Large Prime in dem Sinne nicht so wertvoll sind, daß eben mehr als zwei Large Primes zumindestens in einer weiteren Relation vorkommen müssen, ist die Verwendung von mehr als zwei Large Primes mit diesen Algorithmen nicht sinnvoll.

Der laufzeitintensivste Teil der Relationenberechnung ist, wie schon erwähnt, die Vielzahl der Divisionen bei der Probedivision. Jedes Paar, das das Sieben mit Logarithmen überlebt, erfordert zunächst einmal eine Division durch jedes rationale Faktorbasiselement. Da aber nur sehr wenige Faktorbasiselemente wirklich teilen, sind fast alle Divisionen in gewissem Sinne überflüssig. Sie sind allerdings nur dann unnötig, wenn man sich auf einem anderen Weg die teilenden Faktorbasiselemente beschaffen kann. Dies kann aber dadurch realisiert werden, daß man anstelle des Probedivisionsalgorithmus die Siebalgorithmen 2.13 RATSIEB und 2.18 ALGSIEB verwendet. Das entsprechende Bitarray $H_b[a]$ wird dabei so besetzt, daß gilt

$$H_b[a] = \begin{cases} 1, & \text{falls für } (a,b) \text{ Probedivision durchgeführt werden soll} \\ 0, & \text{sonst.} \end{cases}$$

Durch diese Vorgehensweise wird die Anzahl der Divisionen drastisch reduziert. Für ein Faktorbasiselement muß eine Division ausgeführt werden, um den Startwert für das Sieben zu berechnen. Weitere Divisionen werden nur dann benötigt, wenn das Faktorbasiselement den entsprechenden Wert eines zu untersuchenden Paares teilt. Nach Durchführung der Siebalgorithmen müssen noch die Reste untersucht werden, d.h. der Algorithmus LPCHECK wird durchgeführt. Da im modifizierten Probedivisionsalgorithmus nicht nur über den Faktorbasen zerlegt wird, sondern auch die entsprechenden Exponentenvektoren aufgebaut werden, müssen der RATSIEB und der ALGSIEB Algorithmus dahingehend abgewandelt werden, daß auch die Exponentenvektoren berechnet werden.

Beispiel 3.23 (Laufzeit ohne Probedivision)
Betrachtet man wieder Z_{89} aus Beispiel 3.22, so läßt sich eine Laufzeit von 70.58 Sekunden für die Relationensuche messen. Man benötigt also nur 7.8% der Laufzeit des ursprünglichen Verfahrens. Das Berechnen einer Relation kostet nun nur noch 3.07 Sekunden im Vergleich zu den 39.13 Sekunden aus Beispiel 3.22. Verwendet man den neuen Algorithmus auch, wenn man nur eine Large Prime pro Zerlegung zuläßt, so reduziert sich die Laufzeit von 50.52 Sekunden auf 32.98 Sekunden, also um etwa 35%. Dabei werden allerdings nur 3 Relationen generiert, so daß die Zeit, die benötigt wird, um eine Relation zu erzeugen, etwa 11 Sekunden beträgt. Man braucht also in diesem Beispiel in der Double Large Prime Variante etwa 3.6 mal solange, um eine Relation zu generieren, wie in der Quadruple Large Prime Variante.
□

Bemerkung 3.24 (weitere Verbesserungen)

1. Es ist sinnvoll, Probedivision zuerst mit der algebraischen Faktorbasis durchzuführen, da dort die größeren Werte zerlegt werden müssen. Die Wahrscheinlichkeit, daß ein Paar $(a.b)$ diesen Schritt überlebt, ist deshalb geringer als die Wahrscheinlichkeit, daß dieses Paar das Probedividieren mit der rationalen Faktorbasis überlebt. Durch diese Vorgehensweise reduziert sich die Anzahl der Paare, für die unnötigerweise mit beiden Faktorbasen probedividiert werden muß. In obigem Beispiel läßt sich die Gesamtlaufzeit auf 61.84 Sekunden drücken, was 2.69 Sekunden pro Relation entspricht. Die Anzahl der Paare, für die mit beiden Faktorbasen Probedivision durchgeführt werden mußte, verringert sich von 311 auf 110.

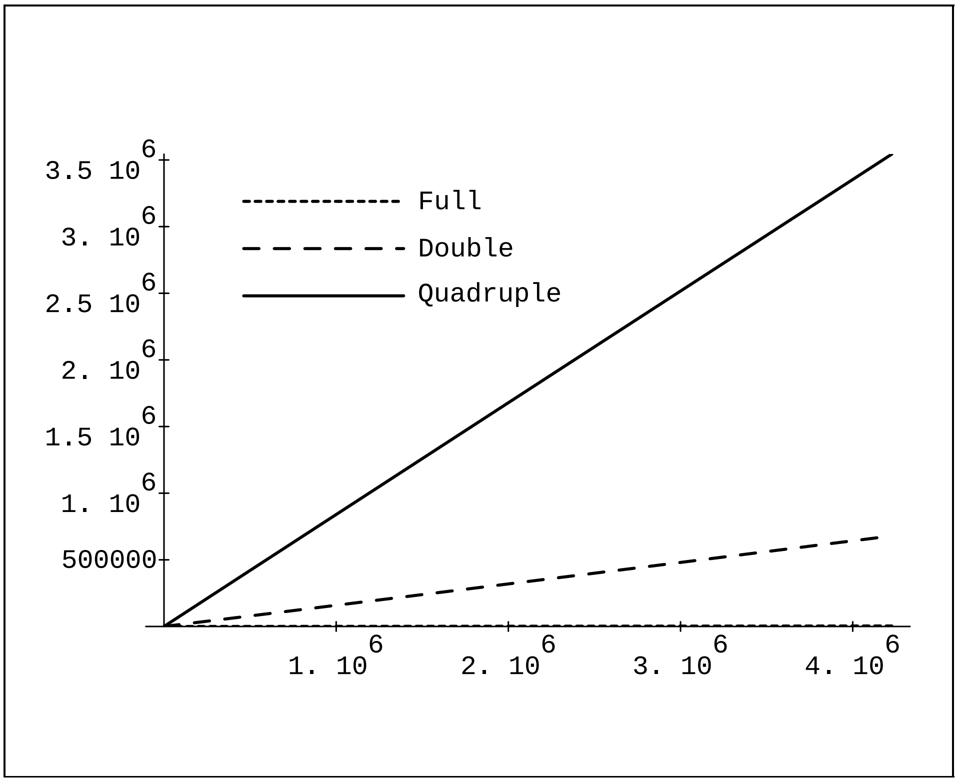
2. Auch beim Probedivisionssieb kann man auf kleine Primzahlen verzichten, da diese den Siebschritt zu zeitaufwendig machen. Für diese muß dann allerdings der modifizierte Probedivisionsalgorithmus durchgeführt werden.

3. Anstatt im Probedivisionssieb die Werte $a+b\,m$ bzw. $|N(a,\,b)|$ durch Primzahlen zu dividieren, kann man die Siebeinträge auch mit 1 initialisieren und die Primzahlen aufmultiplizieren. Erst nachdem mit allen Faktorbasiselementen gesiebt wurde, wird durch das Produkt dividiert. Bei dieser Vorgehensweise muß man allerdings im modifizierten Probedivisionsalgorithmus auch die höheren Potenzen der teilenden Faktorbasiselemente herausdividieren. □

Anzahl der Zykel

Die Quadruple Large Prime Variante nimmt, wie oben gezeigt, pro Relation weniger Zeit in Anspruch als die Double Large Prime Variante. Allerdings ist die Qualität der Relationen schlechter, da mehr Large Primes erlaubt sind. Die folgenden beiden Bilder zeigen die Entwicklung der Zykel in Abhängigkeit von der Anzahl der gesiebten Relationen am Beispiel der Faktorisierung von Z_{80}. Abbildung 3.4 schlüsselt die gesiebten Relationen in solche ohne Large Primes (Full), mit einer oder zwei Large Primes (Double) bzw. drei oder vier Large Primes (Quadruple) auf. Abbildung 3.5 zeigt, wieviele Full Relationen gefunden wurden, wieviele Zykel nur Relationen mit maximal zwei Large Primes enthalten (Double) und die Anzahl der Zykel mit Relationen mit bis zu vier Large Primes (Quadruple).

Abbildung 3.4: Verteilung der Relationen im Beispiel Z_{80}



Man kann erkennen, daß zu einem gewissen Zeitpunkt, im Beispiel etwa bei 3.75 Millionen Relationen, die Anzahl der Zykel mit mindestens einer Relation mit mehr als zwei Large Primes sehr stark ansteigt. Genauer habe ich dieses Phänomen am Beispiel Z_{30} (siehe Kapitel 7.3) untersucht. Es ist zwar nicht effizient, für Zahlen dieser Größenordnung die Quadruple Large Prime Variante zu verwenden, dennoch lassen sich auch an solch kleinen Beispielen wertvolle Erkenntnisse sammeln. Unter der Nutzung von extrem kleinen Faktorbasen, $|FB_R| = 150$ und $|FB_A| = 500$, bei Large Prime Grenzen von 50000 im rationalen und 400000 im algebraischen Fall, läßt sich eine ähnliche Entwicklung der Zykel feststellen (siehe Abbildung 3.6).

Abbildung 3.5: Entwicklung der Zykel im Beispiel Z_{80}

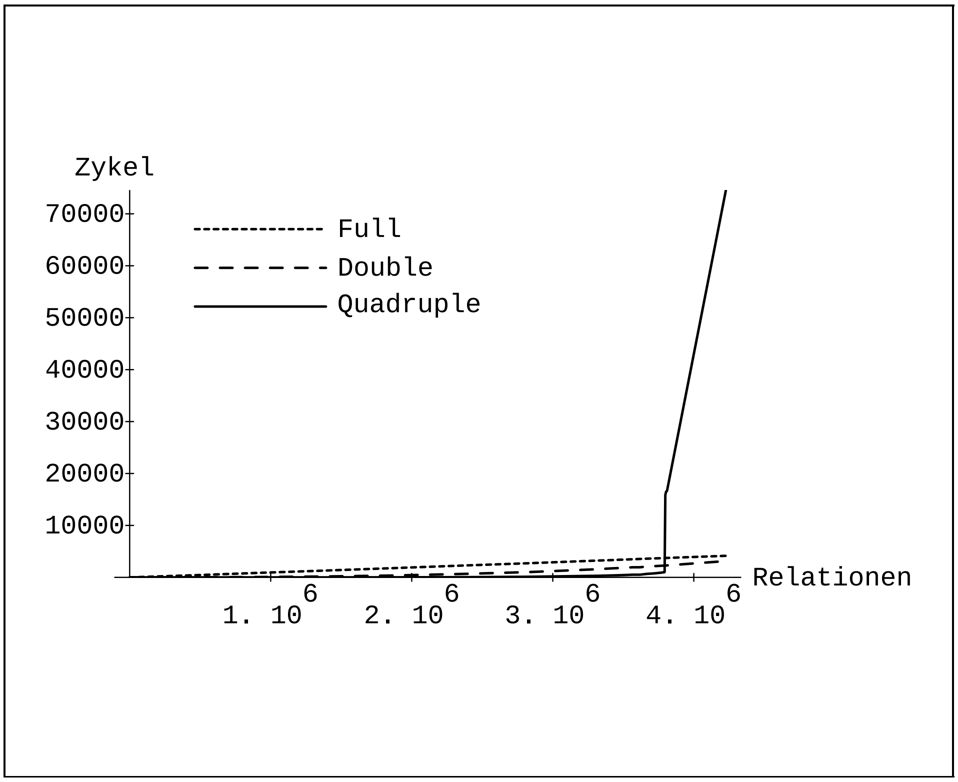
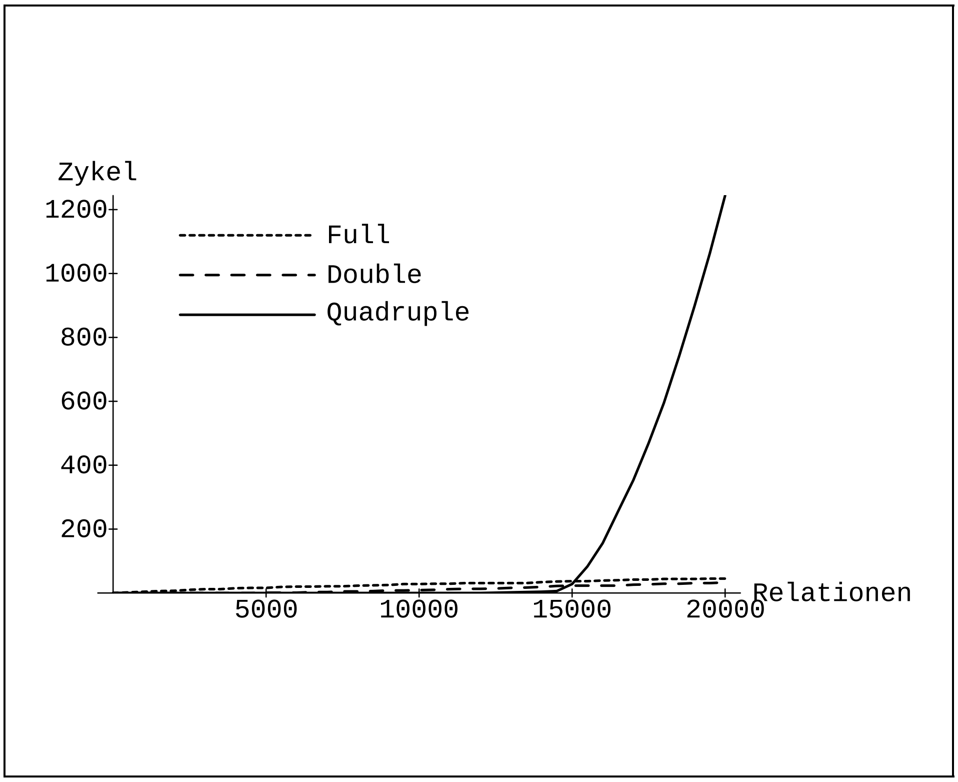
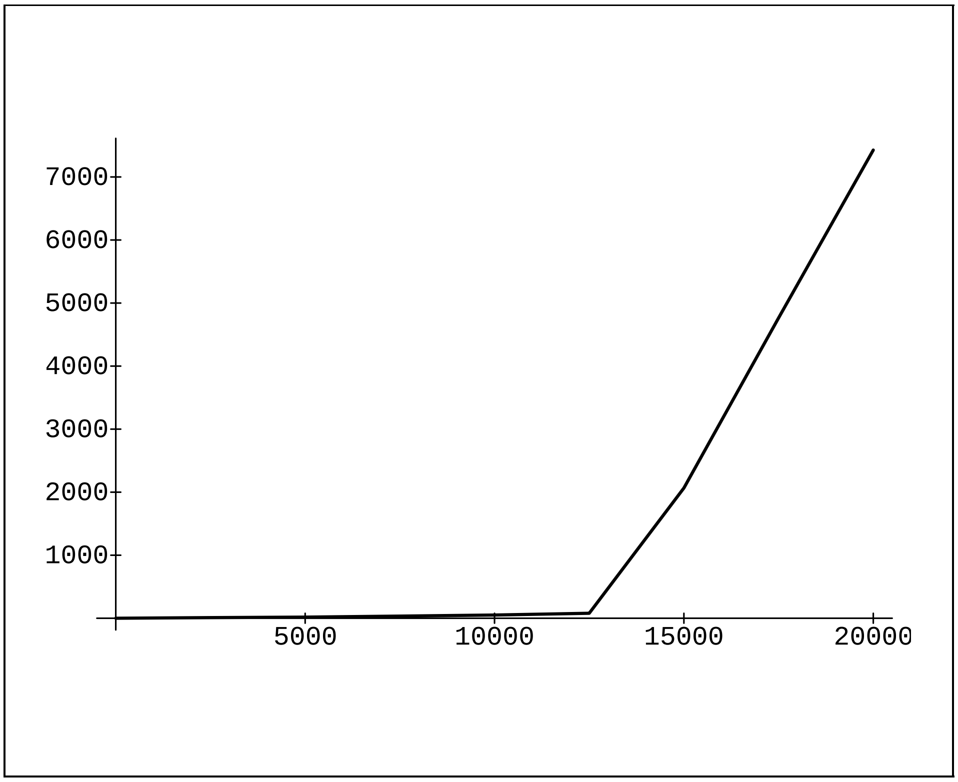


Abbildung 3.6: Entwicklung der Zykel im Beispiel Z_{30}



Eine Erklärung für den explosiven Anstieg der Zykel läßt sich sehr leicht finden, wenn man die Anzahl der Relationen betrachtet, die den RELREDUCE–Algorithmus von Seite 61 überstehen.
Die entsprechende Kurve (Abbildung 3.7) zeigt ein ähnliches Verhalten wie die Kurve der Zykelentwicklung. Eine genauere Untersuchung der Stelle, an der dieses starke

Abbildung 3.7: Überlebende des RELREDUCE–Algorithmus



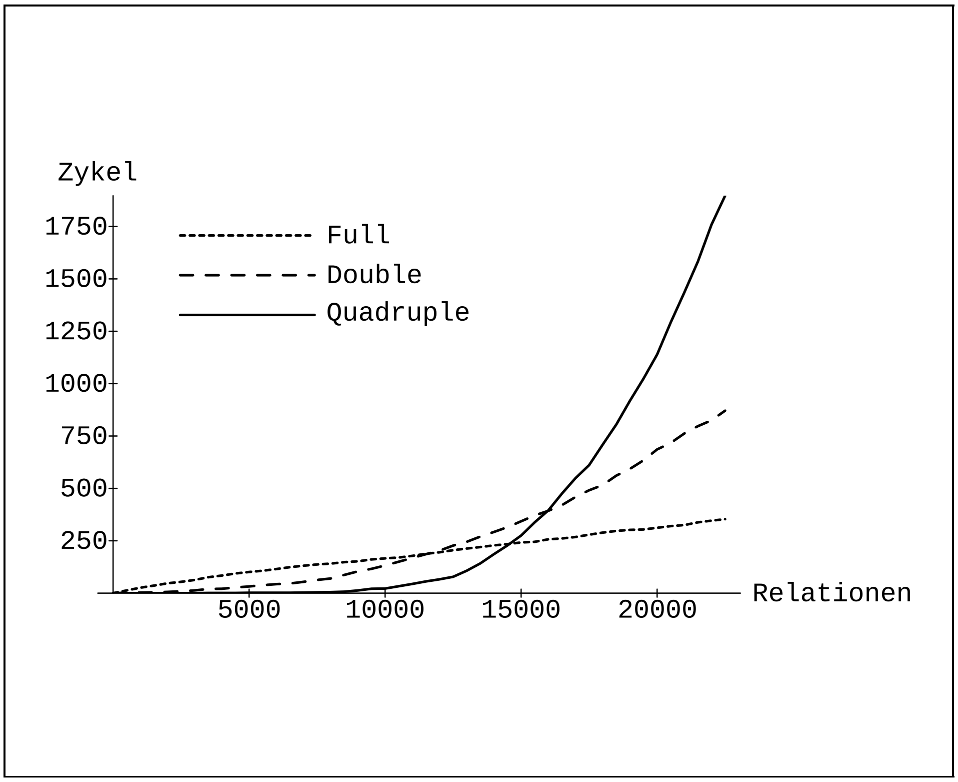
Ansteigen festzustellen ist, ergibt, daß durch Hinzunahme einer einzigen Single Relation eine Art Kettenreaktion ausgelöst wird. Die Large Prime dieser Single macht das Überleben einer Triple Relation möglich, deren beiden anderen Large Primes in insgesamt 23 anderen Relationen vorkommen. Von diesen überstehen fünf zum ersten Mal den Reduktionsschritt. Diese fünf Relationen erlauben ihrerseits wiederum anderen Relationen das Überleben, und so pflanzt sich dies fort. Tabelle 3.5 zeigt diese Explosion in Zahlen.
Die Anzahl der Zykel ändert sich allerdings nach Einfügen der 14385. Relation zunächst noch nicht. Dafür erhöht sich aber die Anzahl der Zusammenhangskomponenten von 1 auf 136. Die nächsten generierten Relationen bewirken auf der einen Seite eine weitere Erhöhung der Anzahl dieser Komponenten bis zum maximalen Wert von 309, liefern aber auf der anderen Seite nur einen leichten Zuwachs an berechneten Zykeln. Nach dem Einfügen von etwa 1000 weiteren Relationen können die Zusammenhangskomponenten zu einer verschmolzen werden. Aufgrund der großen Anzahl von Knoten in dieser Komponente lassen sich die weiteren Relationen, die den RELREDUCE–Algorithmus passiert haben, direkt in den Graphen einbauen

Tabelle 3.5: Explosion in Zahlen

Anzahl der Relationen					
vor Reduktion	nach Reduktion				
total	total	Single	Double	Triple	Quadruple
14384	109	59	37	12	1
14385	1069	177	394	371	127

oder werden als Teil eines Zykels herausgeschrieben.
Verwendet man größere Faktorbasen, so bleibt die eben beschriebene Explosion aus. Man kann im Gegenteil ein langsames Ansteigen der Anzahl der Zykel erkennen. Abbildung 3.8 zeigt die Zykelentwicklung für die 30–stellige Zahl bei Faktorbasisgrößen von $|FB_R| = 300$ und $|FB_A| = 1500$.

Abbildung 3.8: Zykelentwicklung mit großer Faktorbasis



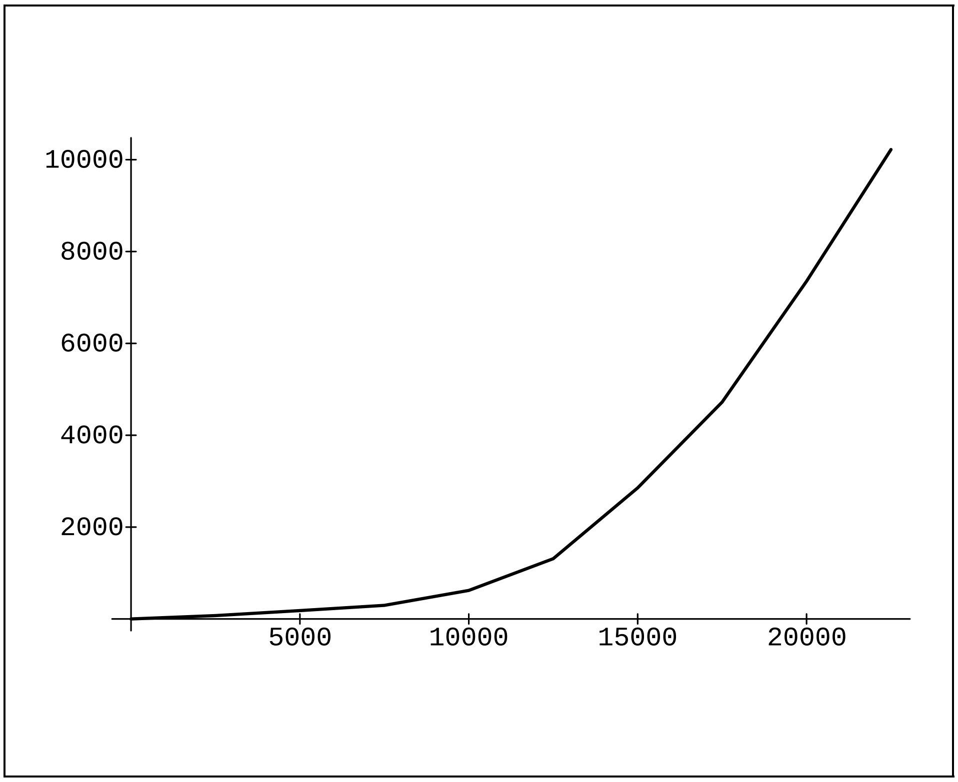
Der Grund für dieses unterschiedliche Verhalten ist in dem unterschiedlichen Prozentsatz, mit dem die verschiedenen Relationentypen vorkommen, zu suchen (Tabelle 3.6).
Im Beispiel mit den großen Faktorbasen kann ein wesentlich höherer Anteil von Single und Double Relationen festgestellt werden. Die Wahrscheinlichkeit, daß eine

Tabelle 3.6: Relationenverteilung bei verschiedenen Faktorbasen in Prozent

$\lvert FB_R \rvert$	$\lvert FB_A \rvert$	Full	Single	Double	Triple	Quadruple
150	500	0.22	3.14	18.76	44.66	33.21
300	1500	1.57	12.84	34.59	37.72	13.28

Relation den RELREDUCE–Algorithmus übersteht, ist natürlich umso höher, je weniger Large Primes in der Relation vorkommen. Demzufolge überleben bei großen Faktorbasen schon zu einem früheren Zeitpunkt wesentlich mehr Relationen den Reduktionsschritt, was durch Abbildung 3.9 bestätigt wird.

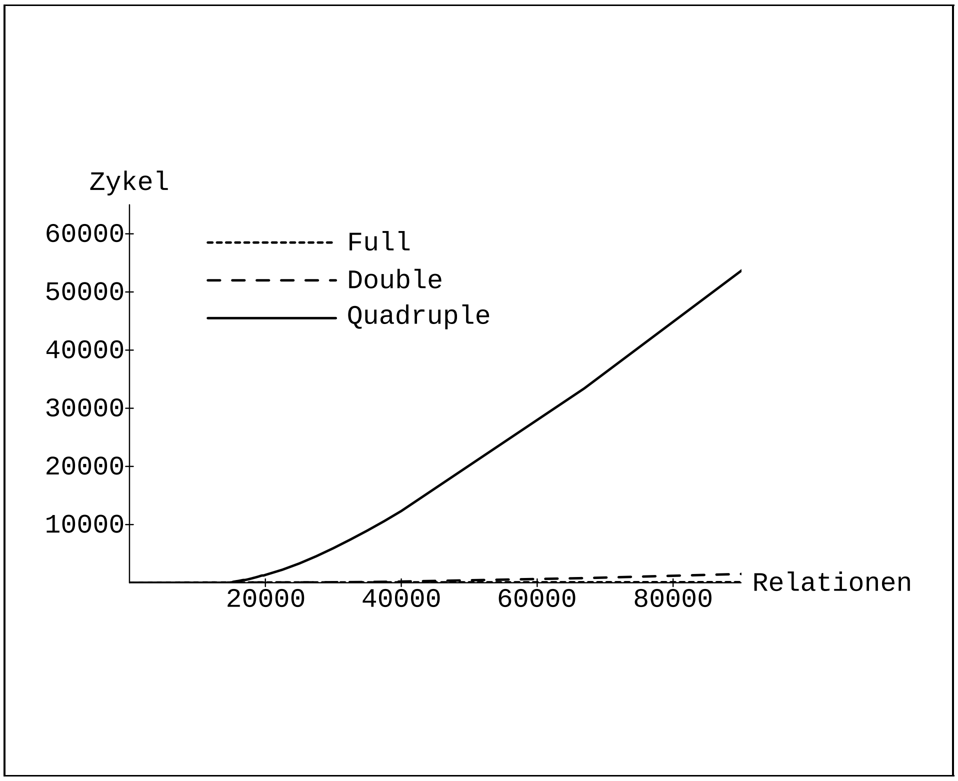
Abbildung 3.9: Reduktionsschritt mit großer Faktorbasis



Diese überlebenden Relationen sind zunächst einmal nur Single und Double Relationen. Alle Single Relationen liegen in einer Komponente, nämlich in der mit der Wurzel $v_0 = (1, 0)$. Baut man die Double Relationen mit ein, so zeigt sich, daß diese bis auf ganz wenige Ausnahmen ebenfalls in dieser Komponente liegen. Dies führt dazu, daß schon zu einem sehr frühen Zeitpunkt Zykel gefunden werden. Überstehen Relationen mit mehr als zwei Large Primes den Reduktionsschritt, so können sie in den meisten Fällen komplett in Graph aufgenommen werden.
Unabhängig von der Größe der Faktorbasis läßt sich aber erkennen, daß es die meisten Zykel vom Quadruple–Typ sind.

Betrachten wir einmal, wie sich die Entwicklung der Zykel verhält, wenn man immer neue Relationen generiert, auch wenn schon bei weitem genug Zykel gefunden worden sind, um die Zahl zu faktorisieren. Siebt man lange genug, so wird man alle vorkommenden Large Primes in den Graphen einbauen können, d.h. alle kommen mindestens zweimal vor. Weiter werden irgendwann alle Large Primes in der gleichen Zusammenhangskomponente liegen. Dann liefert jede gefundene Relation einen Zykel. Die Kurve verläuft dann linear. Dies wird durch Abbildung 3.10. noch einmal dokumentiert. Dort ist die Anzahl der Zykel in Abhängigkeit von der Anzahl der Relationen für Z_{30} aufgeführt, wobei am Ende fast 100 mal so viele Zykel wie benötigt gefunden wurden.

Abbildung 3.10: Fast lineare Zykelentwicklung am Beispiel Z_{30}



Dabei lieferten die letzten 1000 generierten Relationen 925 Zykel.

Vergleich der Varianten

Auf der einen Seite kann man durch Verwendung von vier Large Primes pro Relation die Relationensuche beschleunigen, auf der anderen Seite bleiben aber negative Seiteneffekte nicht aus. So erhöht sich beispielsweise das Gewicht der Matrix (siehe Seite 35), d.h. die Anzahl der Matrixeinträge ungleich 0, enorm. Dies ist eine Folge der Tatsache, daß sehr viele Relationen miteinander kombiniert werden müssen, damit die Large Primes nur in gerader Potenz vorkommen. Die Anzahl der Relationen, die in einem Zykel vorkommen, werde ich im weiteren mit *Zykellänge* bezeichnen. Full

Relationen sind dabei Zykel der Länge 1. Je größer die Zykellänge ist, umso dichter wird natürlich auch die entsprechende Spalte in der Matrix. Tabelle 3.7 zeigt am Beispiel der Faktorisierung von Z_{65} (vergleiche Kapitel 7.8), wie sich die durchschnittliche Zykellänge bei der QLP bzw. DLP Variante verhält. Weiter wird dort aufgeführt, inwieweit sich die entsprechenden Matrizen in Dimension und Gewicht unterscheiden.

Tabelle 3.7: Auswirkung der Large Prime Variante auf die Matrix

Variante	durchschnittliche Zykellänge	Dimension der Matrix	Gewicht der Matrix
DLP–Variante	2.616	41920×34736	1369923
QLP–Variante	20.059	27485×24756	4674597

Die vergrößerte Anzahl von Relationen pro Matrixspalte bringt natürlich auch eine vergrößerte Anzahl von Paaren (a, b) in der Menge S mit sich. Dies führt zu verlängerten Laufzeiten in der Berechnung der algebraischen Quadratwurzel (siehe Kapitel 5). Diese negative Nebenwirkungen der QLP–Variante kann man dadurch reduzieren, daß man etwas mehr siebt. Sind die Parameter so eingestellt, daß eine Zykelexplosion stattfindet, so lassen sich durch geringen Siebaufwand einerseits viele neue Zykel und andererseits wesentlich kleinere Zykel berechnen.

Tabelle 3.8: Die Zykellänge in Abhängigkeit von der Anzahl der Relationen am Beispiel Z_{30}

Anzahl der Relationen	längster Zykel	durchschnittliche Zykellänge
18000	4803	642.97
19000	409	163.69
20000	199	88.15
21000	116	56.43
22000	88	42.97
23000	62	31.97
24000	50	26.31
25000	40	21.71
26000	36	19.74
27000	30	17.16

Für die Zahl Z_{30} habe ich bei Faktorbasen von $|FB_R| = 150$ und $|FB_A| = 500$ die QLP– Variante angewandt. In Tabelle 3.8 ist festgehalten, wie sich die durchschnittliche Zykellänge der 700 kürzesten Zykel in Abhängigkeit von der Gesamtanzahl an Relationen ändert. Die Tabelle beginnt bei 18000 Relationen, da etwa bei dieser Anzahl zum ersten Mal 700 Zykel erreicht wurden.
Betrachtet man die Zykellängen nach 18000 Relationen, so sieht man, daß die durchschnittliche Länge fast schon der Anzahl der Faktorbasiselemente entspricht. In diesem Falle ist die Matrix nicht mehr dünnbesetzt. Aber schon nach kurzem zusätzlichen Sieben reduziert sich die Durchschnittslänge und somit das Gewicht der entsprechenden Matrix stark.
Das gleiche Verhalten der Zykellängen läßt sich auch an größeren Beispielen feststellen. So findet man beim Faktorisieren der 152–stelligen Zahl Z_{152} mit etwa 140000 Faktorbasiselementen die in Tabelle 3.9 aufgeführten Zykellängen. Die Laufzeit der

Tabelle 3.9: Die Zykellänge in Abhängigkeit von der Anzahl der Relationen am Beispiel Z_{152}

Anzahl der Relationen	Anzahl der Zykel	längster Zykel	durchschnittliche Zykellänge
insgesamt			
4786718	260169	19538	899.23
5598465	622325	3326	237.5
die 150000 kürzesten Zykel			
4786718	150000	844	345.95
5598465	150000	92	44.96

Siebphase erhöhte sich zur Berechnung der weiteren $5598465 - 4786718 = 811747$ Relationen um etwa 25 Prozent.

Trotz dieser Nachteile und dem dadurch notwendigen 'Nachsieben' ist das Faktorisieren 65–stelliger Zahlen mit der Quadruple Large Prime Variante wesentlich schneller als unter Verwendung der Double Large Prime Variante. Dies liegt einfach daran, daß die Siebphase den größten Anteil an der Laufzeit (siehe Tabelle 7.1 von Seite 159) ausmacht.

In Tabelle 3.10 sind Daten der Faktorisierungen der Zahl Z_{65} einerseits mit der Double Large Prime Variante und andererseits mit der Quadruple Large Prime Variante aufgeführt.
Solange die Laufzeit der Siebphase durch die Quadruple Large Prime Variante so signifikant verkürzt werden kann, ist diese Variante der Double Large Prime Variante überlegen. In der Praxis beginnt diese Überlegenheit bei zu faktorisierenden Zahlen von etwa 50 Dezimalstellen.

Tabelle 3.10: Vergleich Double und Quadruple Large Prime Variante am Beispiel Z_{65}

	DLP	QLP
Faktorbasen		
$\|FB_R\|$	9000	6000
$\|FB_A\|$	25722	18752
Laufzeiten in Minuten auf einer 21 Mips Machine		
Berechnung der Faktorbasen	1.94	1.35
Berechnung der Relationen	24181.9	14269.8
Berechnung der Zykel	26.88	90.11
Lösen des LGS	76.45	151.86
Berechnung der Quadratwurzeln	126.42	673.6
total	24413.6	15186.7

3.5 Das Lattice Sieve

Das Lattice Sieve (LS) ist eine Variante der Siebphase des NFS–Algorithmus, die auf eine Idee von J. Pollard ([Po91]) aus dem Jahre 1991 zurückgeht.

3.5.1 Sieben im Gitter

Pollard verfolgte dabei folgenden Ansatz zur Berechnung der Menge T genügend vieler guter Paare. Anstatt sich ein festes $1 \leq b \leq B_{max}$ zu wählen und alle Paare (a, b) mit $A_{min} \leq a \leq A_{max}$ zu untersuchen, nimmt man sich ein rationales Faktorbasiselement $q = p_j \in FB_R$ (im folgenden Special_q genannt) und bestimmt die guten Paare in einem gewissen Bereich, die gerade q als größten Teiler von $a + bm$ besitzen, also

$$a + b \cdot m = \prod_{p_i \in FB_R} {p_i}^{e_{p_i}}$$

mit $e_{p_j} > 0$ und $e_{p_i} = 0, \forall j < i \leq |FB_R|$ gilt. Sei M die Menge aller guten Paare nach Definition 2.7, so läßt sich M durch die disjunkte Vereinigung der Mengen M_q darstellen

$$M = \dot{\bigcup_{q \in FB_R}} M_q,$$

wobei M_q die Menge der guten Paare ist, für die q das größte Faktorbasiselement ist, das $a + bm$ teilt. Im Lattice Sieve Algorithmus werden Teile dieser Mengen M_q berechnet. Dabei wird ausgenutzt, daß die Menge

$$L_q = \{(a, b) \in \mathbb{Z}^2 \mid q | (a + bm)\}$$

ein zweidimensionales Gitter in $\mathbb{Z}^2$ bildet. Seien $\vec{u} = (u_1, u_2)$ und $\vec{v} = (v_1, v_2)$ zwei Basisvektoren des Gitters, so lassen sich alle Punkte von L_q durch

$$\begin{aligned} (a, b) &= c \cdot \vec{u} + d \cdot \vec{v} \\ &= (c \cdot u_1 + d \cdot v_1,\, c \cdot u_2 + d \cdot v_2) \end{aligned}$$

mit $c,\, d \in \mathbb{Z}$ darstellen.
Elemente aus M_q lassen sich nun dadurch berechnen, daß man sich analog zum ursprünglichen Siebverfahren Schranken C_{min}, C_{max} und D_{max} wählt und alle Paare (a, b) untersucht, die sich durch

$$(a, b) = (c \cdot u_1 + d \cdot v_1,\, c \cdot u_2 + d \cdot v_2)$$

mit $C_{min} \leq c \leq C_{max}$ und $1 \leq d \leq D_{max}$ ergeben, also im Gitter liegen.
Die Überprüfung, ob $a + bm$ bzw. $N_f(a, b)$ von einem rationalen bzw. algebraischen Faktorbasiselement geteilt wird, kann gleich behandelt werden, wenn wir wieder auf die Darstellung der rationalen Faktorbasiselemente als Paare $(p,\, cp)$ mit $cp = m \pmod{p}$ zurückgreifen.
Ein Faktorbasiselement $(p,\, cp)$ mit $cp \neq \infty$ teilt genau dann, wenn

$$\begin{aligned} a + b \cdot cp &\equiv (c \cdot u_1 + d \cdot v_1) + (c \cdot u_2 + d \cdot v_2) \cdot cp \pmod{p} \\ &\equiv 0 \pmod{p}. \end{aligned} \tag{3.15}$$

Hieraus folgt durch Umformung

$$c \cdot (u_1 + u_2 \cdot cp) + d \cdot (v_1 + v_2 \cdot cp) \equiv 0 \pmod{p}. \tag{3.16}$$

Ist $(u_1 + u_2 \cdot cp) \not\equiv 0 \pmod{p}$, so muß gelten

$$c \equiv -d \cdot \frac{v_1 + v_2 \cdot cp}{u_1 + u_2 \cdot cp} \pmod{p}. \tag{3.17}$$

Ist hingegen

$$u_1 + u_2 \cdot cp \equiv 0 \pmod{p},$$

so kann nicht gleichzeitig auch

$$v_1 + v_2 \cdot cp \equiv 0 \pmod{p}$$

sein, da die beiden Basisvektoren von L_q sonst nur ein Untergitter, nämlich das von den beiden Faktorbasiselementen q und (p, cp) aufgespannte, erzeugten. Somit kann die Kongruenz (3.16) nur dann erfüllt sein, wenn d ein Vielfaches von p ist.
Zum Berechnen der guten Paare kann man fast analog zu den Algorithmen RATSIEB und ALGSIEB vorgehen. Der Unterschied besteht darin, daß man hier ein $1 \leq d \leq D_{max}$ festhält und alle c im erlaubten Bereich untersucht. Dazu startet man für jedes Faktorbasiselement $(p,\, cp)$ mit der kleinsten Stelle $c_{(p,cp)}$, an der dieses Faktorbasiselement teilt. Diese Stelle läßt sich durch ihren Abstand $r_{(p,cp)}$ zu C_{min} berechnen.

$$r_{(p,cp)} = -(C_{min} \cdot (u_1 + u_2 \cdot cp) + d \cdot (v_1 + v_2 \cdot cp)) \qquad (\text{Mod } p).$$

Für $c_{(p,cp)}$ gilt dann
$$c_{(p,cp)} = C_{min} + r_{(p,cp)}.$$
Die weiteren Stellen, an denen $(p,\ cp)$ teilt, lassen sich durch sukzessives Aufaddieren von p bestimmen, denn aus
$$c \cdot (u_1 + u_2 \cdot cp) + d \cdot (v_1 + v_2 \cdot cp) \equiv 0 \pmod p$$
folgt
$$(c + l \cdot p) \cdot (u_1 + u_2 \cdot cp) + d \cdot (v_1 + v_2 \cdot cp) \equiv 0 \pmod p$$
für alle $l \in \mathbb{Z}$.

Für den Fall $u_1 + u_2 \cdot cp \equiv 0 \pmod p$ muß man nur die d bestimmen, die von p geteilt werden. Die gesamte, einem solche d entsprechende Zeile wird dann von dem Faktorbasiselement geteilt.

Für die Faktorbasiselemente (p, ∞) kommen nur die Paare (a, b) in Frage, für die b ein Vielfaches von p ist. In diesem Fall muß also nach (3.15) gelten
$$c \cdot u_2 + d \cdot v_2 \equiv 0 \pmod p.$$
$r_{(p,\infty)}$ ergibt sich in diesem Falle unter der Voraussetzung, daß u_2 kein Vielfaches von p ist, zu
$$r_{(p,\infty)} = -d \cdot \frac{v_2}{u_2} \qquad (\mathrm{Mod}\ p)$$
und $c_{(p,\infty)}$ zu
$$c_{(p,\infty)} = C_{min} + r_{(p,\infty)}.$$
Wenn $p|u_2$, so muß
$$d \cdot v_2 \equiv 0 \pmod p$$
sein, also $d \equiv 0 \pmod p$, da der Fall $v_2 \equiv 0 \pmod p$ hier nicht auftreten kann.

3.5.2 Berechnung der Gitterbasis

Im letzten Abschnitt wurde aufgeführt, wie man im Gitter L_q nach guten Paaren sieben kann. Nun möchte ich besprechen, wie man Basisvektoren dieses Gitters berechnet. Dabei wird berücksichtigt, daß es günstig ist, bezüglich der Euklidischen Norm möglichst kurze Vektoren zu finden. Man ist nämlich bemüht, möglichst solche Paare (a, b) mit betragsmäßig kleinen Werten a und b zu betrachten, da diese Paare zu kleineren Werten $a + bm$ bzw. $|N_f(a, b)|$ führen. Deshalb sollte man versuchen möglichst alle Paare, die in
$$L_q \cap AB$$
liegen (AB ist das Rechteck, in dem mit dem ursprünglichen NFS gesiebt wurde), durch das Lattice Sieve zu erwischen. Kleine Linearkombinationen von langen Basisvektoren führen aber auch zu großen Paaren (a, b), die nicht in AB liegen. Deshalb werden besser möglichst kurze Basisvektoren genommen. Diese lassen sich dadurch bestimmen, daß man sich zuerst zwei beliebige Basisvektoren berechnet und dann auf diese den Gaußschen Basisreduktionsalgorithmus für die Ebene (siehe [Va91]) anwendet.

Zwei beliebige Basisvektoren sind dabei schnell gefunden. Stellen wir das spezielle Faktorbasiselement wieder als Paar (q, cq) dar, so bilden die Vektoren

$$(-cq, 1) \text{ und } (q, 0) \tag{3.18}$$

eine Basis von L_q (siehe [GLM94, Kapitel2]).

Beispiel 3.25 (Basisvektoren)
Bei der Anwendung des LS–Algorithmus für RSA130 (siehe Kapitel 7.13) wurde für $(q, cq) = (9739703, 8421252)$ eine Basis berechnet. Als erste Basis ergab sich nach (3.18) $\{(-8421252, 1), (9739703, 0)\}$. Durch Anwendung des Gaußalgorithmus konnte diese Basis zu

$$\{(731, 1736), (-4783, 1965)\}$$

verkürzt werden. Verwendet man $C_{max} = -C_{min} = 4000$ und $D_{max} = 2000$, so werden also insgesamt etwa $16 \cdot 10^6$ Paare untersucht. Diese liegen bei der Ausgangsbasis im Bereich

$$\begin{array}{rcl} -33675268297 & \leq a \leq & 53164414000 \\ 0 & \leq b \leq & 4000 \end{array}$$

Dabei wurde, wenn ein Paar (a, b) mit $b < 0$ gefunden wurde, stattdessen das Paar $(-a, -b)$ betrachtet. Nutzt man dagegen die verkürzte Basis, so ergibt sich der folgende Bereich, in dem die Paare

$$\begin{array}{rcl} -12490000 & \leq a \leq & 2919217 \\ 0 & \leq b \leq & 10874000 \end{array}$$

liegen. □

3.5.3 Vergleich mit dem ursprünglichen Sieb

Den Geschwindigkeitsvorteil zieht das Lattice Sieve gegenüber der ursprünglichen Siebvariante daraus, daß weniger Paare untersucht werden müssen, um genügend viele Relationen zu erhalten.
Wurden im normalen NFS t Paare untersucht, so teilte das Faktorbasiselement (q, cq) durchschnittlich t/q dieser Paare. Im Lattice Sieve sollten deshalb ebensoviele Stellen für q untersucht werden. Insgesamt beläuft sich die Anzahl dieser Stellen auf

$$s = \sum_{q \in FB_R} \frac{t}{q}.$$

Für eine Faktorbasis von 100000 Elementen ist s etwa $2.9 \cdot t$. Das heißt, wenn man LS für alle Faktorbasiselemente durchführt, muß man fast dreimal soviele Paare untersuchen wie im normalen Sieb.
Berücksichtigt man aber, daß es im untersuchten Bereich nur sehr wenige Paare gibt, für die $a + bm$ nur das Produkt von Potenzen sehr kleiner Faktorbasiselemente

ist, so erkennt man, daß man darauf verzichten kann, kleine Special_q zuverwenden. Verzichtet man auf die kleinsten 1000 Faktorbasiselemente, so verringert sich die Anzahl der betrachteten Paare schon auf

$$s = 0.449 \cdot t.$$

Tabelle 3.11 zeigt die Summe der Quotienten $1/q$ bei insgesamt 100000 Faktorbasiselementen.

Tabelle 3.11: Summe $1/q$

Special_q ab Index	$1/q$	Special_q ab Index	$1/q$
95001	0.004	90001	0.008
85001	0.013	80001	0.017
75001	0.022	70001	0.028
65001	0.034	60001	0.040
55001	0.047	50001	0.055
45001	0.064	40001	0.073
35001	0.085	30001	0.098
25001	0.113	20001	0.133
15001	0.159	10001	0.197
5001	0.266	1	2.906

Würde man nur für die größten 5000 Special_q sieben, so wäre der Laufzeitgewinn enorm, aber auch der Verlust an Relationen wäre riesengroß. In der Praxis sollte man deshalb die Special_q erst ab etwa einem Zehntel der Faktorbasisgröße, also

$$\text{Special\_q} > p_{\frac{B1}{10}}$$

verwenden.

Bemerkung 3.26 (Varianten)

1. Anstatt die Special_q aus der rationalen Faktorbasis zu wählen, kann man auch auf die algebraische Faktorbasis zurückgreifen. Die Berechnungen gehen dann analog. Der Vorteil dieser Variante liegt in der Tatsache, daß in der Regel die algebraische Faktorbasis mehr Elemente aufweist als die rationale. Dadurch müssen zwar mehr Gitter aufgebaut werden, man kann sich aber auf relativ kleine Werte für C_{max} und D_{max} zurückziehen und auf diese Weise gewährleisten, daß nur relativ kleine Paare (a, b) untersucht werden (siehe auch Bemerkung 3.27).

2. Die für den normalen NFS–Siebalgorithmus gültigen Varianten wie Sieben mit Logarithmen oder Quadruple Large Prime Variante finden analog im Lattice Sieve Anwendung.

□

Bemerkung 3.27 (Die Laufzeit)
D. J. Bernstein und A. K. Lenstra zeigten in [BeLe93, Kapitel 6.5], daß das Lattice Sieve asymptotisch um den Faktor log (MAXP) schneller ist als das normale Sieb, wobei MAXP das größte Faktorbasiselement der Faktorbasis bezeichnet, aus der die Special_q genommen werden. □

Der Cross–Over–Point der beiden Varianten bei der Verwendung meiner Implementierungen liegt etwa bei 85 Dezimalstellen, wie Tabelle 3.12 bestätigt. Zur Ermittlung dieses Wertes habe ich für die in der Tabelle aufgeführten Beispiele mit bis zu 80 Dezimalstellen die Siebphase mit beiden Varianten vollständig durchgeführt. Für die größeren Zahlen wurden nur für Bruchteile der Faktorbasis die Siebphase ($\sqrt{B_{max}}$ verschiedene b bzw. $\sqrt{B2}$ verschiedene Special_q) durchgeführt.

Tabelle 3.12: Laufzeitvergleich NFS und LS

n	Laufzeit(LS)/Laufzeit(NFS)
Z_{65}	1.19
Z_{70}	1.10
Z_{80}	1.03
Z_{89}	0.98
Z_{107}	0.91
Z_{130}	0.69

3.6 Verteiltes Sieben

Die Relationensuche ist der laufzeitintensivste Schritt des NFS Algorithmus. Das Sieben für eine 80–stellige Zahl (siehe Kapitel 7.11) dauert beispielsweise 325 Tage auf einem Rechner mit 21 MIPS. Eine Beschleunigung der Siebphase ist daher wünschenswert.
Die benötigte Rechenzeit läßt sich zwar nicht weiter reduzieren, allerdings kann man durch Verteilen der Siebphase auf mehrere Rechner die Zeit, die der Anwender warten muß, bis eine Zahl faktorisiert ist, erheblich verkürzen.
Die Vorgehensweise im Relationensuche–Algorithmus 2.20 von Seite 31 war so, daß ein $b \in \{1, \ldots, B_{max}\}$ fest gewählt wurde und für alle $a \in \{A_{min}, \ldots, A_{max}\}$ die Bedingungen von Definition 3.13 untersucht wurden.
Hieraus ergibt sich direkt, wie man die Relationensuche verteilen kann, nämlich durch Verteilen der verschiedenen b auf die einzelnen Rechner.
In der Praxis verwende ich hierzu die C–Bibliothek LiPS, die am Lehrstuhl Prof. Buchmann an der Universität des Saarlandes unter der Leitung von R. Roth ([Ro92])

entstanden ist und von Th. Setz ([Se91]) weiterentwickelt wurde. Diese Bibliothek erlaubt in einfacher Weise, Programme auf Rechnernetzen zu verteilen, wobei lediglich dann auf den Rechnern gearbeitet wird, wenn sie nicht von anderen Anwendern belegt sind. LiPS unterstützt dabei das Client–Server Modell. Dies bedeutet, daß ein ausgezeichneter Rechner, der sogenannte Server, die einzelnen Aufgaben auf die anderen Rechner des Netzes, den sogenannten Clients, verteilt. Die Clients bearbeiten ihre Teilaufgabe und senden die Ergebnisse zurück zum Server, der diese weiterverarbeitet.
Für den Fall des NFS bedeutet dies, daß der Server die Menge der zu untersuchenden b in kleine disjunkte Teilmengen aufteilt und jedem Client eine solche Teilmenge zusendet. Die Clients führen für alle b dieser Teilmenge die Relationensuche durch und senden abschließend die gefundenen Relationen zum Server. Dieser sammelt die Relationen und läßt im Gegenzug dem Client eine neue zu bearbeitende Menge zukommen.
Auch der Lattice Sieve Algorithmus läßt sich leicht verteilen. Anstatt die b zu versenden, müssen nun die speziellen Faktorbasiselemente, für die das Gitter aufgebaut wird, an die einzelnen Clients verteilt werden.
Unter der Verwendung von 80 Rechnern konnte auf diese Art die Laufzeit der Relationensuche für Z_{80} auf 4.5 Tage reduziert werden. Wie lange dabei auf den einzelnen Clients gerechnet wurde, hing einzig und allein davon ab, wie stark der jeweilige Rechner von anderen Benutzern in Beschlag genommen war.
Tabelle 3.13 demonstriert, die unterschiedliche Ausnutzung der verschiedenen Rechner im Netz am Beispiel Z_{80}.

Tabelle 3.13: Verteilte Berechnung der Relationen mit LiPS

Rechnername	Anzahl der Relationen	Laufzeit in Stunden
cip27.cscip.uni-sb.de	133285	104.3
gipsy20.cs.uni-sb.de	125174	95.9
gipsy19.cs.uni-sb.de	107748	79.3
gipsy18.cs.uni-sb.de	98121	75.1
gipsy3.cs.uni-sb.de	56325	45.9
cip20.cscip.uni-sb.de	47415	33.4

Kapitel 4

Wahl des Polynoms

Gegenstand dieses Kapitels ist die Bestimmung der ganzen Zahl m und des irreduziblen, möglicherweise nichtnormierten Polynoms $f(x)$, das die Eigenschaft (2.4) besitzt. Wie wir sehen werden, gibt es unendlich viele solcher Polynome. Allerdings existieren Polynome, bei deren Verwendung die Siebphase wesentlich weniger Zeit benötigt als bei anderen. Die Fragen, welches Polynom zur kürzesten Laufzeit des NFS–Algorithmus führt, und wie man es bestimmen kann, zählen zu den ungelösten Problemen in Verbindung mit dem *Number Field Sieve* . In diesem Zusammenhang werde ich ein Maß angeben, das die Wahrscheinlichkeit beschreibt, daß ein gegebenes Paar (a, b) gut ist. Damit ist es möglich, Polynome gegeneinander abzuwägen. Davon ausgehend werde ich abschließend einen einfachen probabilistischen Algorithmus zur Berechnung eines guten Polynoms vorstellen.

4.1 Berechnung von m und $f(x)$

Ausgangspunkt der Berechnung des Polynoms $f(x)$ und einer Wurzel m modulo n, ist der Grad D des Polynoms. Dieser ergibt sich durch die Formel 2.44 zu

$$D = \left(3^{1/3} + o(1)\right) \cdot (\log n / \log\log n)^{1/3}.$$

Tabelle 4.1 zeigt in der ersten Spalte, welcher Grad für Zahlen bestimmter Größenordnungen verwendet würde, wenn man in obiger Formel $o(1)$ durch 0 ersetzt. Die zweite Spalte gibt die Werte an, wie ich sie in der Praxis verwende. Dabei muß berücksichtigt werden, daß durch die Algorithmen zum Berechnen der algebraischen Quadratwurzel (siehe Kapitel 5) nur ungerade Grade in der Praxis möglich sind.
Hat man den Grad des Polynoms anhand der Tabelle bestimmt, so ergibt sich m aus

$$m := \left\lceil \sqrt[D+1]{n} \right\rceil . \tag{4.1}$$

Diesen Wert kann man mit Hilfe des Algorithmus A.2, den ich auf Seite 162 beschreiben werde, berechnen.
In Abhängigkeit von m läßt sich nun ein Polynom, das die Bedingung (2.4) erfüllt, durch die Berechnung der m-adischen Darstellung generieren. Diese Darstellung ist folgendermaßen definiert:

Tabelle 4.1: Einsatzbereich der Polynomgrade

Polynomgrad	Größenordung von n	
2	bis 10^{13}	-
3	10^{14} bis 10^{42}	bis 10^{75}
4	10^{43} bis 10^{98}	-
5	10^{99} bis 10^{190}	10^{76} bis 10^{200}
6	10^{191} bis 10^{329}	-

Definition 4.1 (m-adische Darstellung)
Seien $z \in \mathbb{N}$, $m \in \mathbb{N}_{\geq 2}$, dann heißt ein Tupel

$$(z_0, \ldots, z_k) \text{ mit } z_i \in \{0, \ldots, m-1\}, z_k \neq 0,$$

m-adische Darstellung der Länge $k+1$ von z, wenn gilt

$$z = z_0 + z_1 \cdot m + \ldots + z_k \cdot m^k.$$

□

Bemerkung 4.2 (Eindeutigkeit)
Diese Darstellung von $z \in \mathbb{N}$ ist eindeutig. □

Die kleinste Zahl, die man mittels m-adischer Darstellung der Länge $k+1$ beschreiben kann, ist

$$m^k \cong (0, \ldots, 0, 1), \tag{4.2}$$

die größte dagegen

$$(m-1, \ldots, m-1) \cong (m-1) \cdot \sum_{i=0}^{k} m^i = (m-1) \cdot \frac{m^{k+1}-1}{m-1} = m^{k+1} - 1. \tag{4.3}$$

Berechnet man die m-adische Darstellung $(f_0, \ldots, f_k)$ von n, so folgt aus (4.1), (4.2) und (4.3), daß $k = D$ sein muß, und es gilt nach Konstruktion für $f(x) = f_0 + \ldots + f_D \cdot x^D$ die Identität

$$f(m) = n.$$

Ausgehend von $f(x)$ lassen sich leicht weitere Polynome generieren, die (2.4) erfüllen, denn es gilt für $r \in \mathbb{Z}$, $0 < i \leq D$ und $\tilde{f}(x) := f(x) + r \cdot (x^i - m \cdot x^{i-1})$

$$\tilde{f}(m) = f(m) + r \cdot (m^i - m \cdot m^{i-1}) = f(m) = n. \tag{4.4}$$

Wählt man m groß genug, so läßt sich auf diese Art und Weise ein Polynom erzeugen, dessen Koeffizienten zwischen $\lfloor -m/2 \rfloor$ und $\lfloor m/2 \rfloor$ liegt. Die größte Zahl, die durch ein solches Polynom darstellbar ist, beläuft sich auf

$$z = \frac{m}{2} + \frac{m}{2} \cdot m + \ldots + \frac{m}{2} \cdot m^D.$$

Will man n so darstellen, dann muß m mindestens

$$\lceil \sqrt[D+1]{2 \cdot n} \rceil \tag{4.5}$$

betragen.
Der folgende Algorithmus berechnet ein solches Polynom.

Algorithmus 4.3 (modifizierte m–adische Dartstellung)

MADIC

EINGABE: $n, m, D \in \mathbb{N}$
AUSGABE: Polynom $f(x) \in \mathbb{Z}[x]$ mit $f(m) = n$ und $|f_i| \leq m/2$

PHASE 1: m–ADISCHE DARSTELLUNG

```
(1)  for (i = 0; i ≤ D; i ++) do
(2)    f_i = n Mod m              /* Divisionsrest */
(3)    n = ⌊n/m⌋                  /* ganzzahliger Anteil */
(4)  od
```

PHASE 2: MODIFIKATION

```
(5)  for (i = 0; i < D; i ++) do
(6)    if (f_i > m/2) then        /* Koeffizient zu groß? */
(7)      f_i = f_i − m            /* verschiebe Koeffizient in den
                                     gewünschten Bereich */
(8)      f_{i+1} = f_{i+1} + 1    /* korrigiere den Fehler */
(9)    fi
(10) od
```

Beispiel 4.4 (modifizierte m–adische Darstellung)
Wählt man für $n = 391$ den Grad $D = 3$, so ergibt sich für $m = 7$ die m–adische Darstellung

$$(6, 6, 0, 1)$$

von $391 = 1 \cdot 343 + 6 \cdot 7 + 6$. Phase 2 des MADIC–Algorithmus erzeugt hieraus die modifizierte m–adische Darstellung

$$(-1, 0, 1, 1)$$

also $1 \cdot 343 + 1 \cdot 49 - 1 = 391$. Es ergibt sich aus n und m also das Polynom

$$f(x) = x^3 + x^2 - 1$$

wie es in Beispiel 2.10 auf Seite 23 schon benutzt wurde.

Legt man sich nicht darauf fest, daß $f(m) = n$ sein muß, sondern erlaubt man auch $f(m) = k \cdot n$, für $k \in \mathbb{N}$, was aufgrund der Bedingung

$$f(m) \equiv 0 \pmod{n} \tag{4.6}$$

möglich ist, so ergibt sich ein weiterer Weg, Polynome zu erzeugen. Dazu nutzt man die Tatsache aus, daß

$$L := \{(f_0, \dots, f_D) \mid \sum_{i=0}^{D} f_i \cdot m^i \equiv 0 \pmod{n}\}$$

eine Untergruppe von Z^{D+1} ist. Eine Basis von L besteht aus dem Vektor $(n, 0, \dots, 0)$ sowie den D Vektoren $(0, \dots 0, -m, 1, 0, \dots, 0)$. Jede Linearkombination dieser Vektoren führt natürlich zu einem Polynom, das (4.6) erfüllt. Im Hinblick auf schnelles Faktorisieren interessieren aber vor allem Polynome mit betragsmäßig kleinen Koeffizienten, da die Koeffizienten die Größe der zu zerlegenden Normen mitbestimmen. Deshalb wendet man auf die Basis den Gitterbasisreduktionsalgorithmus LLL ([LLL82]) an. Mindestens einer dieser Basisvektoren $(f_0, \dots, f_D)$ wird die Eigenschaft

$$\sum_{i=0}^{D} f_i \cdot m^i \neq 0 \tag{4.7}$$

besitzen. Wählt man einen Vektor mit betragsmäßig kleinen Einträgen, der (4.7) erfüllt, so hat man ein Polynom mit den gewünschten Eigenschaften erzeugt.

Beispiel 4.5 (Polynomberechnung mit LLL)
Zum Faktorisieren von Z_{89} könnte man $m = 7445445343435436$ wählen. Der MADIC Algorithmus liefert das Polynom

$$\begin{array}{rrrlrrl} f(x) = & & 633480879 & \cdot x^5 & + & 1703947430724167 & \cdot x^4 \\ & + & 709744250069188 & \cdot x^3 & + & 2537342601967842 & \cdot x^2 \\ & + & 2344638491772085 & \cdot x & - & 2686540804855993 & \end{array}$$

Die LLL–Variante generiert dagegen das Polynom

$$\begin{array}{rrrlrrl} l(x) = & & 156007969608570 & \cdot x^5 & + & 92710506746337 & \cdot x^4 \\ & + & 151326298786339 & \cdot x^3 & + & 8590716657563 & \cdot x^2 \\ & - & 156712245311935 & \cdot x & + & 73555669461729. & \end{array}$$

Für dieses Polynom gilt $l(m) = 246271 \cdot n$.

4.2 Bewertung von Polynomen

4.2.1 Vergleich aus der Praxis

Durch die Wahl des Parameters m und durch die Möglichkeit, das mit Algorithmus 4.3 berechnete Polynom unter Nutzung von (4.4) zu verändern, lassen sich unendlich viele Polynome erzeugen, die den für das NFS notwendigen Zahlkörper definieren. Allerdings wirkt sich das Polynom sehr stark auf die Anzahl der Relationen aus, die in der Siebphase gefunden werden. Dies möchte ich an einem kleinen Beispiel verdeutlichen. Für eine 50–stellige Zahl wurde mit verschiedenen Werten für m die Siebphase des NFS Algorithmus durchgeführt. Auf der rationalen Seite wurde eine Large Prime, auf der algebraischen Seite zwei Large Primes zugelassen. Die Siebparameter 'Größe der Faktorbasen', 'Large Prime Grenzen', 'Siebgröße' usw. blieben bis auf das algebraische Polynom stets die gleichen. Dennoch lassen sich extreme Unterschiede in der Anzahl der generierten Relationen feststellen, wie Tabelle 4.2 zeigt.

Tabelle 4.2: Auswirkung des Parameters m bzw. des Polynoms auf die Anzahl der Relationen

m	Relationen vom Typ					
	ffff	pfff	ffpf	pfpf	ffpp	pfpp
2251294393669	1789	3095	5933	9745	5146	8417
2251294393670	1190	2023	4037	6722	3795	6189
6753883180983	2020	3575	6269	10980	5517	9377
9005177574646	1982	3339	5879	10025	4925	8254
11256471968310	2983	5411	8946	15981	7750	13179
13507766361970	3394	6150	9420	17217	4834	8463
15759060755629	2663	4742	7916	14423	7204	12654

4.2.2 Berechnung der Güte eines Polynoms

Will man eine Zahl schnell mit dem NFS faktorisieren, so bedeutet dies, daß möglichst schnell möglichst viele Relationen gefunden werden müssen. In diesem Zusammenhang nenne ich ein Polynom $f(x)$ *besser* als ein Polynom $g(x)$, wenn bei Verwendung ansonsten gleicher Parameter durch f mehr Relationen erzeugt werden als durch g.
Je besser das Polynom ist, umso schneller läßt sich damit faktorisieren. Allerdings weiß man erst nach dem Sieben, welches von zwei Polynomen man hätte nehmen müssen. Da das Sieben ein sehr kostenintensives Unterfangen ist, möchte man im voraus entscheiden können, welches Polynom das bessere ist.
Leider sieht man den Polynomen ihre *Güte* nicht direkt an, so gehört z.B. zu

$m = 2251294393669$ aus Tabelle 4.2 das Polynom

$$314083334840 - 27675604372 \cdot x - 431262262301 \cdot x^2 + 2251294393634 \cdot x^3$$

und zu $m = 2251294393670$ das Polynom

$$-89503323053 + 834848920121 \cdot x - 431262262190 \cdot x^2 + 2251294393631 \cdot x^3.$$

Die beiden Wurzeln m unterscheiden sich nur um 1, und auch die Koeffizienten der Polynome weisen in etwa die gleiche Anzahl an Dezimalstellen auf. Dennoch liefert das Sieben mit dem ersten Polynom ungefähr 1.5–mal soviele Relationen wie mit dem zweiten.

Im folgenden möchte ich ein Verfahren angeben, das ohne Durchführung der Siebphase das bessere von zwei gegebenen Polynomen bestimmt.

Ein teilerfremdes Paar $(a, b) \in \mathbb{Z}^2$ ist nach Definition 3.13 genau dann eine Relation, wenn $a + bm$ über der rationalen und $N_f(a, b)$ über der algebraischen Faktorbasis bis auf je zwei Large Primes zerfällt. Betrachtet man für vorgegebene Faktorbasisgrößen und ein festes Siebintervall den Logarithmus Φ_R des durchschnittlichen Wertes von $|a + bm|$ und den Logarithmus Φ_A von $|N_f(a, b)|$ sowie die durchschnittlichen Werte R_R und R_A, die durch das Sieben mit Logarithmen (siehe Kapitel 3.2) abgezogen werden, so gibt der Wert

$$M_f = \Phi_R - R_R + \Phi_A - R_A \tag{4.8}$$

ein Maß für die Güte des verwendeten Polynoms an. Je größer M_f ist, umso kleiner ist die Wahrscheinlichkeit, daß ein Paar (a, b) des Siebintervalls eine Relation ist und umgekehrt.

Im folgenden möchte ich erläutern, wie die 4 Teilsummen, aus denen sich M_f zusammensetzt, berechnet werden können. Dazu sei das zugrundeliegende Siebintervall gleich $\{(a, b) \in \mathbb{Z}^2 | -A_{max} \leq a \leq A_{max},\ 1 \leq b \leq B_{max}\}$ mit $A_{max}, B_{max} \in \mathbb{N}$, wobei $A_{max} < m$, so daß $a + bm$ im untersuchten Intervall stets positiv ist.

1. Φ_R

Für festes $b > 0$ gilt nun wegen $b \cdot m > |a|$

$$|-a + b \cdot m| + |a + b \cdot m| = 2 \cdot |b \cdot m|$$

und für den Durchschnitt $\Phi_{R,b}$ der Werte $|a + b \cdot m|$:

$$\begin{aligned} \Phi_{R,b} &= \left(\sum_{a=-A_{max}}^{A_{max}} |a + b \cdot m| \right) / (2 \cdot A_{max} + 1) \\ &= \left(\sum_{a=1}^{A_{max}} 2 \cdot b \cdot m + b \cdot m \right) / (2 \cdot A_{max} + 1) \\ &= b \cdot m. \end{aligned}$$

Somit ergibt sich als Gesamtdurchschnittswert

$$\begin{aligned}
\Phi_{R,B_{max}} &= \left(\sum_{b=1}^{B_{max}} \Phi_{R,b}\right) / (B_{max}) \\
&= \left(\sum_{b=1}^{B_{max}} b \cdot m\right) / (B_{max}) \\
&= \frac{B_{max} \cdot (B_{max}+1)}{2} \cdot m/(B_{max}) \\
&= \frac{B_{max}+1}{2} \cdot m. && (4.9) \\
& && (4.10)
\end{aligned}$$

Φ_R ist also $\log\left(\frac{B_{max}+1}{2} \cdot m\right)$.

2. Φ_A

Für festes b entspricht die Summe der Normen $N_f(a,b)$ in etwa der Fläche A_b, die vom Normpolynom

$$N_b(a) = f\left(\frac{-a}{b}\right) \cdot (-b)^D$$

und der x–Achse in den Grenzen $-A_{max}$ und A_{max} eingeschlossen wird. Um diese Fläche berechnen zu können, benötigt man auf der einen Seite die Stammfunktion $\tilde{N}_b(a)$ des Normpolynoms und auf der anderen Seite die reellen Nullstellen von $N_b(a)$. Die Stammfunktion läßt sich sehr leicht bestimmen: sei

$$N_b(a) = a_0 + a_1 \cdot x + \ldots + a_D \cdot x^D$$

so gilt für die Stammfunktion

$$\tilde{N}_b(a) = c + a_0 \cdot x + \frac{1}{2} \cdot a_1 \cdot x^2 + \ldots + \frac{1}{D+1} \cdot a_D \cdot x^{D+1}, \ c \in \mathbb{R}.$$

Aus den reellen Nullstellen $\rho_1, \ldots, \rho_r$ von $f(x)$ kann man die Nullstellen des Normpolynoms $N_b(a)$ herleiten. Diese sind nämlich

$$\xi_1 = -b \cdot \rho_1, \ldots, \xi_r = -b \cdot \rho_r,$$

denn es gilt für $i = 1, \ldots, r$

$$\begin{aligned}
N_b(\xi_i) &= N_b(-b \cdot \rho_i) \\
&= f\left(\frac{-(-b \cdot \rho_i)}{b}\right) \cdot (-b)^D \\
&= f(\rho_i) \cdot (-b)^D \\
&= 0.
\end{aligned}$$

Geht man o.B.d.A. davon aus, daß die reellen Nullstellen ξ_i des Normpolynoms der Größe nach sortiert sind und alle im Intervall $[-A_{max}, A_{max}]$ liegen (wenn nicht betrachtet man nur die Nullstellen in diesem Intervall), so gilt für die eingeschlossene Fläche A_b

$$\begin{aligned} A_b &= \Big|\int_{-A_{max}}^{\xi_1} N_b(a)\,da\Big| + \sum_{i=1}^{r-1}\Big|\int_{\xi_i}^{\xi_{i+1}} N_b(a)\,da\Big| + \Big|\int_{\xi_r}^{A_{max}} N_b(a)\,da\Big| \\ &= |\tilde{N}_b(-A_{max}) - \tilde{N}_b(\xi_1)| + \sum_{i=1}^{r-1}|\tilde{N}_b(\xi_{i+1}) - \tilde{N}_b(\xi_i)| + |\tilde{N}_b(\xi_r) - \tilde{N}_b(A_{max})|. \end{aligned}$$

Φ_A ergibt sich dann zu

$$\Phi_A = \log\left(\frac{\sum\limits_{b=1}^{B_{max}} A_b}{B_{max}}\right) \tag{4.11}$$

3. R_R

Betrachten wir zunächst einmal ein Faktorbasiselement p und um wieviel dieses in der Siebphase durchschnittlich reduziert. An jeder Stelle (a, b), an der p teilt, wird im LOGSIEB–Algorithmus $\log p$ abgezogen. Nach Definition 3.13 kommen aber nur solche Paare (a, b) in Betracht, die keinen gemeinsamen Primteiler besitzen, also inbesondere nicht kongruent $(0, 0)$ mod p sind. Modulo p gibt es p^2 verschiedene Paare, wovon $p^2 - 1$ inkongruent 0 modulo p sind. Genau $p - 1$ davon entsprechen Stellen, die von p geteilt werden. Somit beläuft sich die Wahrscheinlichkeit, daß p an einer bestimmten Stelle im Siebarray teilt, die keinem Paar kongruent $(0, 0)$ modulo p entspricht, auf

$$W_p = \frac{p-1}{p^2-1} = \frac{1}{p+1}.$$

Der Wert, um den im Mittel durch p reduziert wird, ist somit

$$\frac{\log p}{p+1}.$$

Die Wahrscheinlichkeit, daß p^i an einer bestimmten Stelle teilt, ist um den Faktor p^{i-1} geringer als die Wahrscheinlichkeit, daß p teilt. Im Mittel reduziert also p^i um

$$\frac{\log p^i}{(p+1)\cdot p^{i-1}}$$

reduziert. Insgesamt ergibt sich als Durchschnittsreduktionswert durch das Faktorbasiselement p und seine Potenzen unter Berücksichtigung, daß an den Stellen, wo p^i teilt, auch alle kleineren Potenzen teilen und somit den Reduktionswert jeweils um $\log p$ erhöht haben:

$$\begin{aligned} \sum_{i=1}^{\infty}\frac{\log p}{(p+1)\cdot p^{i-1}} &= \frac{\log p}{p+1}\cdot\sum_{i=1}^{\infty}\frac{1}{p^{i-1}} \\ &= \frac{\log p}{p+1}\cdot\frac{p}{p-1} \end{aligned}$$

$$= \frac{p \cdot \log p}{p^2 - 1}.$$

R_R erhält man durch Aufsummieren der Reduktionswerte aller rationalen Faktorbasiselemente:

$$R_R = \sum_{p \in FB_R} \frac{p \cdot \log p}{p^2 - 1}. \tag{4.12}$$

4. R_A

R_A läßt sich analog zu R_R berechnen. Der algebraische Reduktionsfaktor beträgt hier also

$$R_A = \sum_{\pi_{p,cp} \in FB_A} \frac{p \cdot \log p}{p^2 - 1}. \tag{4.13}$$

4.2.3 Berechnung der Güte in der Praxis

Die Berechnung der Güte eines Polynoms erfordert u.a. die Bestimmung der Wurzeln des Polynoms, die Erstellung der Faktorbasen sowie die Ermittlung der Fläche unter dem Normpolynom. Dies alles ist nur mit relativ großem Rechenaufwand zu realisieren. In den folgenden Abschnitten werde ich angeben, wie man gute Approximationen für die Güte eines Polynoms mit wesentlich weniger Rechenaufwand berechnen kann. Mit $\log(x)$ bezeichne ich dabei den Logarithmus zur Basis 10 von x.

Abschätzen der Durchschnittswerte Φ_R und Φ_A

Anstatt die Durchschnittswerte Φ_R und Φ_A mittels der Formeln (4.9) und (4.11) zu berechnen, kann man sie auch auf probabilistische Art und Weise approximieren. Dazu wählt man sich einige zufällige Paare (a, b) im zulässigen Bereich, berechnet die Werte $|a+b \cdot m|$ und $|N_f(a, b)|$, bildet den jeweiligen Durchschnitt und logarithmiert.

Berechnung der Reduktionswerte

Anhand der Formeln (4.12) und (4.13) sieht man, daß kleine Faktorbasiselemente einen erheblich höheren Beitrag zum Gesamtreduktionswert beisteuern als die großen. Beispielsweise ist

$$\frac{2 \cdot \log 2}{2^2 - 1} = 0.200687$$

während für $p_{10000} = 104729$

$$\frac{104729 \cdot \log 104729}{104729^2 - 1} = 0.0000479$$

ist. Schaut man sich Tabelle 4.3 an, wo Zwischenergebnisse der Berechnung des Reduktionswerts für eine rationale Faktorbasis von 200000 Elementen aufgeführt sind, so erkennt man, daß schon nach 1000 Primzahlen, also nach einem Zweihundertstel

Tabelle 4.3: Zwischenergebnisse der Reduktionswertbestimmung für eine rationale Faktorbasis von 200000 Elementen

Index des Faktor–basiselements	Zwischen–ergebnis R_R	Index des Faktor–basiselements	Zwischen–ergebnis R_R
100	2.253	1000	3.406
2000	3.746	3000	3.943
4000	4.082	5000	4.190
6000	4.278	7000	4.352
8000	4.416	9000	4.473
10000	4.523	20000	4.854
30000	5.047	40000	5.184
50000	5.289	60000	5.375
70000	5.448	80000	5.511
90000	5.566	100000	5.616
150000	5.806	200000	5.942

der zu untersuchenden Faktorbasiselemente, mehr als die Hälfte des Reduktionswertes erreicht wurde. Gleiches kann man auch für den algebraischen Reduktionswert beobachten. Geht man davon aus, daß das Verhältnis zwischen dem Reduktionswert, der durch kleine Faktorbasiselemente erzeugt wird, und dem, der durch große zustande kommt, eine Konstante ist, so kann man sich bei der Berechnung des Maßes eines Polynoms auf kleine Faktorbasiselemente beschänken. Tabelle 4.4 zeigt am Beispiel einer 89–stelligen Zahl, wie sich das Maß von 10 verschiedenen Polynomen, gegeben durch den Parameter m, ändert, wenn man nur Bruchteile der Faktorbasen (original: $|FB_R| = 20000$, $|FB_A| = 60000$) zur Berechnung heranzieht. Vergleicht man die berechneten Werte für das Maß, so sieht man, daß schon nach Verwendung von nur einem Zehntausendstel der Faktorbasen, also 2 rationalen und 6 algebraischen Faktorbasiselementen, die besten sieben der zehn untersuchten Polynome schon in der richtigen Reihenfolge ihrer Güte feststehen. Nach einem Tausendstel sind es dann acht. Für die beiden letzten Polynome wechselt dagegen ihre *Plazierung*. Dies liegt daran, daß sie das gleiche Maß 56.48 aufweisen. Berechnet man die Güte der Polynome für die Parameter aus Tabelle 4.2, so ergeben sich die Plazierungen aus Tabelle 4.5.

Die Plazierung, die sich aufgrund der Approximation der Polynomgüte ergibt, entspricht bis auf eine Ausnahme genau der Reihenfolge, die sich durch Berechnen der Relationen ergeben hat. Lediglich für $m = 9005177574646$ wurde eine Güte vorhergesagt (Platz 3), die durch die Relationengenerierung (Platz 5) nicht bestätigt werden konnte. Dennoch kann man mit der Vorhersage sehr zufrieden sein, denn das bei weitem beste Polynom weist auch das beste Maß auf.

Auf der anderen Seite zeigt die Tabelle aber auch, daß es 'Ausreißer' gibt, die sich beim Sieben deutlich besser oder schlechter verhalten, als es vorhergesagt wurde.

Tabelle 4.4: Maßbestimmung mit verschiedenen Faktorbasen bei Polynome für eine 89–stellige Zahl

m	Bruchteil der Faktorbasen				
	$1/10000$	$1/1000$	$1/100$	$1/10$	$1/1$
1975109341293301	63.48	61.05	58.67	56.45	54.24
1975109341294101	64.22	62.09	59.85	57.58	55.37
1975109341293401	64.33	62.41	59.92	57.66	55.46
1975109341294201	64.55	62.45	60.09	57.80	55.60
1975109341293601	64.91	62.56	60.15	57.92	55.71
1975109341293501	65.00	62.62	60.27	58.02	55.81
1975109341294001	65.02	62.68	60.39	58.16	55.95
1975109341293701	65.40	63.05	60.79	58.54	56.34
1975109341293901	65.48	63.20	60.93	58.69	56.48
1975109341293801	65.20	63.28	60.96	58.69	56.48

Tabelle 4.5: Maß und Relationen am Beispiel Z_{50}

Platz	m	Maß	ffff
6	2251294393669	38.98	1789
7	2251294393670	39.16	1190
5	6753883180983	38.41	2020
3	9005177574646	37.77	1982
2	11256471968310	37.75	2983
1	13507766361970	36.84	3394
4	15759060755629	38.25	2663

Um auszuschließen, daß ein Polynom gewählt wird, das wesentlich schlechter ist, als es das Maß angibt, kann man mit den nach der Güteberechnung besten Polynomen für einige $1 \leq b \leq B_{max}$ bzw. für einige Special_q sieben, um zu vergleichen, wie sich die Polynome in der Praxis verhalten.
Bei der Polynomwahl für RSA130 (siehe Kapitel 7.13) standen 15 Polynome zur Auswahl, die aufgrund relativ kleiner Koeffizienten zur Hoffnung Anlaß gaben, zu relativ kleinen Normen zu führen.
Die folgende Tabelle zeigt die Plazierungen der besten fünf Polynome nach der Maßberechnung und ihre Plazierung nach dem Testsieben mit etwa $\sqrt{B2}$ gleichmäßig verteilten Special_q.
Siebt man für alle 15 Polynome, so schiebt sich Polynom 01 mit 15045 Relationen auf

Tabelle 4.6: Maß und Relationen am Beispiel RSA130

Platz nach der Maßberechnung	Polynom–nummer	Platz nach dem Sieben	Anzahl der Relationen
1	14	1	16379
2	04	2	16076
3	12	3	14275
4	09	4	12550
5	11	5	12373

den dritten Platz. In der Tabelle, die sich durch die Güteberechnung ergibt, findet sich Polynom 01 auf Position 11 wieder. Dennoch ist auch in diesem Fall zu erkennen, daß durch die Berechnung der Güte das beste der 15 Polynome vorhergesagt werden konnte.

4.3 Die Generierung guter Polynome

4.3.1 Polynome für allgemeine Zahlen

Für gegebenen Polynomgrad D beträgt der kleinstmögliche Wert, den die Wurzel m annehmen kann, wenn das Polynom der modifizierten m–adischen Darstellung von n entsprechen soll, nach (4.5)

$$m_0 = \lceil \sqrt[D+1]{2 \cdot n} \rceil.$$

Aus jedem größeren m läßt sich dann mit dem Algorithmus MADIC ein Polynom konstruieren. Nach der Laufzeitanalyse sollte m allerdings in der Größenordnung von $\sqrt[D+1]{n}$ liegen. Für eine 100–stellige Zahl bedeutet dies, daß m etwa 17 Dezimalstellen hat. Es ist klar, daß man in der Praxis nicht alle in Frage kommenden 17–stelligen Zahlen untersuchen kann. Allerdings ist es ein offenes Problem, welches m man wählen muß, um ein gutes Polynom zu erhalten.
Die einzige Methode, die derzeit in der Praxis verwendet wird, besteht aus der Wahl einiger zufälliger m aus einem vorgegebenen Intervall, etwa $[m_0, 10 \cdot m_0]$. Für diese wird ein zugehöriges Polynom mittels des Algorithmus 4.3 oder der entsprechenden LLL–Variante generiert und die jeweilige Güte berechnet. Zum Faktorisieren wird schließlich das Polynom mit dem besten Gütewert eingesetzt.

Algorithmus 4.6 (Berechnung eines guten Polynoms)

NFSPOLY

EINGABE: n, D, $anzahl$
AUSGABE: $m \in \mathbb{Z}$, das durch modifizierte m–adische Darstellung zu gutem Polynom führt

PHASE 1: BERECHNUNG DER STARTWERTE

$m_0 := DROOTZ(2 \cdot n,\, D+1) + 1$ /* Integerwurzel mittels Algorithmus A.2 */
$min\_mass := \infty$ /* Initialisierung */

PHASE 2: m, POLYNOM UND GÜTE

for $(i := 0;\, i < anzahl;\, i\,{+}{+})$ **do** /* Teste anzahl viele m */
  Wähle m zufällig zwischen m_0 und $10 \cdot m_0$ /* Wähle m */
  $f := MADIC(n, m, D)$ /* Polynomberechnung mit Algorithmus 4.3 */
  if $(min\_mass > MASS(f,\, i \cdot m))$ **then** /* das Maß M_f aus (4.8) */
    $min\_mass := MASS(f,\, i \cdot m)$
    $min\_m := m$
  fi
od
return (min_m)

In der Regel liefert die LLL-Variante die besseren Polynome, so entspricht dem Polynom $f(x)$ aus Beispiel 4.5 das Maß 62.61 und dem mit LLL konstruierten Polynom $l(x)$ 61.78. Bei einem Test für Z_{89} wurden 1000 verschiedene m zufällig gewählt. Tabelle 4.7 zeigt einen Vergleich der beiden Methoden.

Tabelle 4.7: Vergleich Polynomkonstruktion mit MADIC und LLL

	MADIC	LLL
min. Maß	60.66	60.66
max. Maß	63.79	63.28
durch. Maß	62.28	62.08

Beide Verfahren lieferten dabei übrigens, $m = 1254993803165995$ und das Polynom

$$\begin{array}{rrlrl} f(x) = & 4655635843176 & \cdot x^5 & + \ 622255161027131 & \cdot x^4 \\ - & 26253395587674 & \cdot x^3 & - \ 178846039875702 & \cdot x^2 \\ + & 364117535641325 & \cdot x & - \ 216997134551653 & \end{array}$$

als beste Wahl.

4.3.2 Polynome für spezielle Zahlen

Es gibt zusammengesetzte Zahlen, die eine so spezielle Form haben, daß es sinnvoll ist, diese Form zur Berechnung des Polynoms auszunutzen. So sind z.B. die Cunningham Zahlen (siehe auch [BLSTW83] oder [BrRi92]) von der Gestalt

$$n = b^e \pm 1.$$

Durch Multiplizieren von n mit einer kleinen Potenz b^k läßt es sich erreichen, daß der Exponent von b durch den gewünschten Polynomgrad D teilbar ist:

$$b^{e+k} \pm b^k \equiv 0 \pmod{n}$$

mit $D \mid e+k$. Dann hat man mit der Wahl

$$m = b^{\frac{e+k}{D}}$$

und

$$f(x) = x^D \pm b^k$$

Polynom und Wurzel mit den gewünschten Eigenschaften gefunden. Die Koeffizienten des Polynoms sind aber für betragsmäßig kleine b ebenfalls sehr klein (b^k), während m etwa so groß wie $\sqrt[D]{n}$ ist.
Alternativ zum Multiplizieren mit einer geeigneten Potenz von b kann man ein gutes Polynom durch Ausklammern einer solchen Potenz aus b^e erhalten:

$$b^k \cdot b^{e-k} \pm 1 \equiv 0 \pmod{n},$$

wobei hier D $e-k$ teilen muß. In diesem Fall ergibt sich m zu

$$m = b^{\frac{e-k}{D}}$$

und das Polynom zu

$$f(x) = b^k \cdot x^D \pm 1.$$

Beispiel 4.7 (Das Maß einer Zahl mit spezieller Form)
Für die Cunninghamzahl $13^{136} + 1$ kann man beispielsweise

$$m = 13^{27}$$

und das Polynom

$$f(x) = 13 \cdot x^5 + 1$$

wählen. Als Maß ergibt sich dann 63.77.
Das beste Maß, das der NFSPOLY–Algorithmus bei einer Untersuchung von 500 verschiedenen Polynomen mit ebensovielen verschiedenen Wurzeln m gefunden hat, ist mit 86.13 wesentlich größer. □

Kapitel 5

Berechnung der algebraischen Quadratwurzel

In diesem Kapitel werde ich, aufbauend auf den Ausführungen im Zusammenhang mit dem Basisalgorithmus in Kapitel 2.4, einen für die Praxis relevanten Algorithmus zur Berechnung des Wertes $\psi(\delta)$ vorstellen, der sich ergibt, wenn man den Homomorphismus ψ auf die Wurzel δ des algebraischen Zahlquadrats

$$g'(\omega)^2 \cdot \prod_{(a,b)\in S} (f_D \cdot a + b\omega) \in \mathbb{Z}[\omega]$$

anwendet, vorstellen.
Dabei dient der erste Abschnitt dazu, zwei bekannte Methoden anzugeben, durch die die Größe der Zwischenergebnisse im Laufe der Berechnung relativ klein gehalten werden können. Danach stelle ich ein neues Verfahren vor, das das algebraische Quadrat zuerst stark reduziert, bevor die Wurzel gezogen wird. Im letzten Abschnitt dieses Kapitets gehe ich auf Aspekte der Implementierung ein und beschreibe unter anderem, wie sich die Berechnung der Wurzel verteilt durchführen läßt. Den Abschluß dieses Kapitels bildet eine Untersuchung über die Zufälligkeit der generierten Lösungen der quadratischen Kongruenz (2.1).

5.1 Berechnung modulo vieler Primzahlen

Der Grund, warum der in Kapitel 2.4 angegebene Algorithmus in der Praxis nur schlecht anwendbar ist, besteht in der riesigen Stellenanzahl der Koeffizienten (siehe Beispiel 2.29).
Die erste Verbesserung, die diesbezüglich vorgenommen werden kann, geht auf [BuLePo93, Kapitel 9] zurück. Dort wird vorgeschlagen, das Produkt

$$\gamma = g'(\omega)^2 \cdot \prod_{(a,b)\in S} (f_D \cdot a + b\omega)$$

in $\mathbb{Z}/p\mathbb{Z}[\omega]$ für ein $p \in \mathbb{P}$, modulo dessen $g(x)$ irreduzibel ist, zu berechnen. In diesem Fall ist nämlich $\mathbb{Z}/p\mathbb{Z}[\omega]$ ein Körper mit p^D Elementen und somit isomorph zu $\mathbb{F}_p[x]/(g(x) \bmod p)$. Primzahlen mit diesen Eigenschaften heißen *träge Primzahlen* oder *inert primes*.

In diesem Körper läßt sich das Inverse δ_0 der Wurzel berechnen, wobei die Koeffizienten der Teilprodukte während der Berechnung durch p beschränkt sind. In [LeTi92, Seite 169 bis 198] sind Algorithmen beschrieben, die dies leisten. Eine Alternative hierzu werde ich im Abschnitt 5.1.1 vorstellen.
Für das Inverse der Wurzel gilt nun

$$\delta_0^2 \cdot \gamma \equiv 1 \pmod{p,\, g(x)}.$$

Ausgehend davon lassen sich mit Hilfe einer Iterationsformel die Inversen der Wurzel modulo p^{2^j} für $j = 1, 2, \ldots$ und $g(x)$ berechnen. Die Iterationsformel ([BuLePo93, Seite 72]) lautet

$$\delta_j \equiv \frac{\delta_{j-1} \cdot \left(3 - \delta_{j-1}^2 \cdot \gamma\right)}{2} \pmod{p^{2^j},\, g(x)}. \tag{5.1}$$

Ist p^{2^j} mindestens doppelt so groß wie der Betrag des größten in der Wurzel vorkommenden Koeffizienten, so ist die Quadratwurzel vollständig durch

$$\delta = \delta_j \cdot \gamma \pmod{p^{2^j},\, g(x)}$$

berechnet.
Durch diese Verbesserung müssen nur noch Zahlen in der Größenordnung der Koeffizienten der Wurzel aus dem algebraischen Quadrat betrachtet werden, was aber dennoch viel zu groß ist.
Bevor ich nun auf weitere Verbesserungen eingehe, möchte ich zuerst ein Verfahren vorstellen, daß die Quadratwurzel in einem Körper $\mathbb{Z}/p\mathbb{Z}[\omega]$ berechnet.

5.1.1 Berechnung einer Quadratwurzel in $\mathbb{Z}/p\mathbb{Z}[\omega]$

Sei $p \in \mathbb{P}_{>2}$, $\mathbb{K} = \mathbb{Z}/p\mathbb{Z}[\omega]$ ein Körper mit p^d Elementen und $q \in \mathbb{K}$ ein Quadrat. Die Berechnung der Quadratwurzel r von q läßt sich durch Verallgemeinerung des Algorithmus von D. Shanks zur Berechnung der Quadratwurzel in $\mathbb{Z}/p\mathbb{Z}$ ([Ch93, Kap. 1.5.1]) erreichen.
Dabei nutzen wir die Tatsache aus, daß die Ordnung der multiplikativen Gruppe $\mathbb{K}^*$ von $\mathbb{K}$ $p^d - 1$ beträgt. Aus Satz 1.29 von Seite 14 folgt, daß für alle $x \in \mathbb{K}^*$ die Gleichung

$$x^{p^d-1} = 1$$

erfüllt ist. Für jedes Quadrat $\tilde{q} = \tilde{r}^2 \in \mathbb{K}^*$ gilt somit auch

$$\tilde{q}^{\frac{p^d-1}{2}} = \left(\tilde{r}^2\right)^{\frac{p^d-1}{2}} = \tilde{r}^{p^d-1} = 1. \tag{5.2}$$

Da p ungerade ist, können wir $p^d - 1$ als

$$p^d - 1 = 2^k \cdot u$$

schreiben, wobei $k, u \in \mathbb{N}$ und u ungerade ist. Da es sich bei q um ein Quadrat handelt, ist q^u ebenfalls ein Quadrat, dessen Wurzel ich im weiteren mit r_u bezeichnen möchte. Wäre r_u bekannt, so ließe sich daraus unter Verwendung der Formel

$$r = q^{\frac{u+1}{2}} \cdot {r_u}^{-1} \tag{5.3}$$

die Wurzel r berechnen, denn es gilt

$$\begin{aligned} r^2 &= \left(q^{\frac{u+1}{2}} \cdot {r_u}^{-1}\right)^2 \\ &= q^{u+1} \cdot {r_u}^{-2} \\ &= q^{u+1} \cdot q^{-u} = q. \end{aligned}$$

Es bleibt also die Wurzel r_u von q^u zu berechnen. Dazu bedienen wir uns der Tatsache, daß die Menge G der Elemente aus $\mathbb{K}^*$ mit Zweierpotenzordnung eine Untergruppe von $\mathbb{K}^*$ ist, und daß q^u in G enthalten ist. Da $\mathbb{K}^*$ zyklisch ist, ist auch G zyklisch (Satz 1.31) und somit gibt es erzeugende Elemente von G, die ich im folgenden mit Primitivwurzel bezeichnen möchte. Dies sind genau die Elemente in G, die Ordnung 2^k haben (Satz 1.30). Sei w eine solche Primitivwurzel. Da G von w erzeugt wird und q^u in G liegt, gibt es einen Exponenten e mit

$$w^e = q^u.$$

Da q^u ein Quadrat ist, muß e gerade sein, woraus sich

$$r_u = w^{\frac{e}{2}}$$

ableiten läßt. So verlagert sich das Problem, die Wurzel aus q^u zu ziehen, auf die beiden Probleme

1. Berechne eine Primitivwurzel w von G.
2. Berechne einen Exponenten e mit $w^e = q^u$.

Das erste Problem ist gleichbedeutend mit der Suche eines Nichtquadrates a in $\mathbb{K}^*$, denn Nichtquadrate besitzen gerade die Eigenschaft, daß ihre Ordnung durch 2^k teilbar ist. Das Element $w = a^u$ hat damit genau die Ordnung 2^k. Da $\mathbb{K}^*$ zyklisch ist, gibt es ebenso viele Quadrate wie Nichtquadrate. Durch folgenden probabilistischen Algorithmus läßt sich deshalb sehr schnell ein Nichtquadrat finden.

Algorithmus 5.1 Nichtquadrat in $\mathbb{Z}/p\mathbb{Z}[\omega]$

NOSQUARE

EINGABE: Körper $\mathbb{Z}/p\mathbb{Z}[\omega]$ der Ordnung p^d mit $p \in \mathbb{P}$ ungerade
AUSGABE: Nichtquadrat in $\mathbb{Z}/p\mathbb{Z}[\omega]$

(1) **do**
(2) Wähle $a \in \mathbb{Z}/p\mathbb{Z}[\omega]$ zufällig
(3) $square = a^{\frac{p^d-1}{2}}$
(4) **while** $(square = 1)$ /* $square = 1 \Rightarrow a$ Quadrat */
(5) **return** (a) /* Rückgabe des Nichtquadrates */

Die Berechnung des Exponenten e erweist sich als aufwendiger. Ich werde im folgenden erläutern, wie sich e bitweise bestimmen läßt.
Da w die Ordnung 2^k hat, gibt es einen Exponenten $0 < e < 2^k$ mit $w^e = q^u$. Dieser Exponent kann binär als

$$e = \sum_{i=0}^{k-1} e_i \cdot 2^i$$

mit $e_i \in \{0, 1\}$ dargestellt werden. Für q^u gilt somit

$$q^u = w^{\sum_{i=0}^{k-1} e_i \cdot 2^i}.$$

Potenziert man die Gleichung mit 2^{k-1}, so erhält man

$$\begin{aligned}
(q^u)^{2^{k-1}} &= w^{2^{k-1} \cdot \sum_{i=0}^{k-1} e_i \cdot 2^i} \\
&= w^{e_0 \cdot 2^{k-1}} \cdot \underbrace{w^{2^k \cdot \sum_{i=1}^{k-1} e_i \cdot 2^{i-1}}}_{=1} \\
&= w^{e_0 \cdot 2^{k-1}}. \qquad (5.4)
\end{aligned}$$

Die linke Seite dieser Gleichung läßt sich leicht durch schnelle Exponentiation (siehe Anhang A.1) bestimmen. Ergibt sie 1, so muß $e_0 = 0$ sein, da die Ordnung von $w = 2^k$ ist. Ist die linke Seite ungleich 1, so gilt $e_0 = 1$.

Angenommen wir hätten die ersten j Bits $e_0, \ldots, e_{j-1}$ von e berechnet, so läßt sich das Bit e_j folgendermaßen bestimmen. Aus

$$q^u = w^{\sum_{i=0}^{k-1} e_i \cdot 2^i}$$

folgt durch Division

$$q^u \cdot w^{-\sum_{i=0}^{j-1} e_i \cdot 2^i} = w^{\sum_{i=j}^{k-1} e_i \cdot 2^i}.$$

Potenziert man die Gleichung mit 2^{k-j-1}, so ergibt sich

$$\begin{aligned}
\left(q^u \cdot w^{-\sum_{i=0}^{j-1} e_i \cdot 2^i} \right)^{2^{k-j-1}} &= w^{\left(\sum_{i=j}^{k-1} e_i \cdot 2^i \right) \cdot 2^{k-j-1}} \\
&= w^{e_j \cdot 2^{k-1}} \cdot \underbrace{w^{2^k \cdot \sum_{i=j+1}^{k-1} e_i \cdot 2^{i-j-1}}}_{=1} \\
&= w^{e_j \cdot 2^{k-1}}.
\end{aligned}$$

Analog zu oben folgt, daß man die linke Seite leicht ausrechnen kann und, wenn diese 1 ist, gilt $e_j = 0$, andernfalls gilt $e_j = 1$.

Auf diese Art lassen sich die Bits des Exponenten e bestimmen. Da q ein Quadrat ist, muß die linke Seite der Gleichung (5.4) 1 sein und somit $e_0 = 0$ gelten. Der Exponent e ist also gerade und man kann somit

$$r_u = w^{e/2}$$

berechnen.

Bemerkung 5.2 (Inverse in $\mathbb{K}^*$)
In dem gerade beschriebenen Verfahren muß mit dem Inversen von

$$w^{\sum_{i=0}^{j-1} e_i \cdot 2^i}$$

multipliziert werden. Da die Ordnung von $\mathbb{K}^*$ $p^d - 1$ beträgt, gilt für ein $a \in \mathbb{K}^*$

$$1 = a^{p^d-1} = a^{p^d-2} \cdot a.$$

Hieraus läßt sich

$$a^{-1} = a^{p^d-2}$$

ableiten. □

Unter bestimmten Voraussetzungen kann man die Wurzel aus q auch auf einem einfacheren Weg erhalten. Diesen Fällen sind die nächsten beiden Abschnitte zugedacht.

1. $p^d \equiv 3 \bmod 4$

Gilt diese Kongruenz, so ergibt sich die Wurzel r zu

$$r = q^{\frac{p^d+1}{4}}. \tag{5.5}$$

Der Beweis ist einfach, denn aufgrund der Voraussetzung $p^d \equiv 3 \bmod 4$ ist $\frac{p^d+1}{4}$ eine ganze Zahl. Für r^2 gilt somit

$$r^2 = \left(q^{\frac{p^d+1}{4}}\right)^2 = q^{\frac{p^d+1}{2}} = q^{\frac{p^d-1}{2}} \cdot q = q$$

Die letzte Gleichheit gilt aufgrund (5.2), da es sich bei q um ein Quadrat handelt.

2. $p^d \equiv 5 \bmod 8$

Auch auf dieser Grundlage läßt sich die Wurzel sehr leicht bestimmen. Dazu muß allerdings vorbemerkt werden, daß dieser Fall nur für ungerades d auftreten kann, da 5 kein quadratischer Rest modulo 8 ist. Dann gilt aber, daß auch $p^d \equiv p \equiv 5 \bmod 8$ sein muß, da das Quadrat jeder ungeraden Zahl kongruent 1 modulo 8 ist.
Weiterhin müssen hier zwei zusätzliche Fälle unterschieden werden. Wegen $q^{\frac{p^d-1}{2}} = 1$ und $\frac{p^d-1}{4} \in \mathbb{Z}$ (letzteres ergibt sich aus der Voraussetzung), gilt nun

$$q^{\frac{p^d-1}{4}} = \pm 1.$$

Ist $q^{\frac{p^d-1}{4}} = 1$ erfüllt, so ist

$$r = q^{\frac{p^d+3}{8}} \tag{5.6}$$

die Wurzel von q, da

$$r^2 = \left(q^{\frac{p^d+3}{8}}\right)^2 = q^{\frac{p^d+3}{4}} = q^{\frac{p^d-1}{4}} \cdot q = q.$$

Das Bilden der Potenz $q^{\frac{p^d+3}{8}}$ ist wiederum nur unter der Voraussetzung $p^d \equiv 5 \bmod 8$ möglich, da nur dann $\frac{p^d+3}{8}$ ganzzahlig ist.

Für $q^{\frac{p^d-1}{4}} = -1$ muß noch ausgenutzt werden, daß für $p \equiv 5 \bmod 8$ (siehe [Ch93, Theorem 1.4.7])

$$2^{\frac{p-1}{2}} \equiv -1 \bmod p$$

ist. r ergibt sich dann zu

$$r = 2^{\frac{p-1}{4}} \cdot q^{\frac{p^d+3}{8}}, \tag{5.7}$$

weil

$$r^2 = \left(2^{\frac{p-1}{4}} \cdot q^{\frac{p^d+3}{8}}\right)^2 = 2^{\frac{p-1}{2}} \cdot q^{\frac{p^d+3}{4}} = -1 \cdot q^{\frac{p^d-1}{4}} \cdot q = q.$$

Der folgende Algorithmus berechnet die Quadratwurzel, wen ein Quadrat eingegeben wird und liefert FALSE, wenn es sich bei der Eingabe um ein Nichtquadrat handelt.

Algorithmus 5.3 Quadratwurzel in $\mathbb{Z}/p\mathbb{Z}[\omega]$

SQUAREROOT

EINGABE: $\mathbb{Z}/p\mathbb{Z}[\omega]$ mit $p \in \mathbb{P}$ ungerade und $|\mathbb{Z}/p\mathbb{Z}[\omega]| = p^d$, $q \in \mathbb{Z}/p\mathbb{Z}[\omega]^*$
AUSGABE:

$$\begin{array}{rl} \text{TRUE, } r \text{ mit } q = r^2, & \text{wenn } q \text{ Quadrat ist} \\ \text{FALSE,} & \text{sonst} \end{array}$$

SPEZIALFÄLLE

(1) **if** $(p^d \equiv 3 \bmod 4)$ **then**
(2) $\quad r = q^{\frac{p^d+1}{4}}$ /* nach (5.5) */
(3) **else**
(4) $\quad$ **if** $(p^d \equiv 5 \bmod 8)$ **then**
(5) $\quad\quad$ **if** $(q^{\frac{p^d-1}{4}} = 1)$ **then**
(6) $\quad\quad\quad r = q^{\frac{p^d+3}{8}}$ /* nach (5.6) */
(7) $\quad\quad$ **else**
(8) $\quad\quad\quad r = 2^{\frac{p-1}{4}} \cdot q^{\frac{p^d+3}{8}}$ /* nach (5.7) */
(9) $\quad\quad$ **fi**

Fortsetzung auf Seite 112

ALLGEMEINER FALL

else
  Bestimme $k,\ u \in \mathbb{N}$ mit
    $p^d - 1 = 2^k \cdot u$ und u ungerade
  $w := (NOSQUARE(p,\ d))^u$ /* Algorithmus 5.1 */
  Bestimme Exponent e mit $w^e = q^u$
  if (e ungerade) **then** /* q kein Quadrat */
    return (FALSE)
  $r := q^{\frac{u+1}{2}} \cdot w^{\frac{-e}{2}}$ /* nach (5.3) */
  fi
fi

TESTE ERGEBNIS

if $(r \cdot r = q)$ **then**
  return (TRUE, r) /* $q = r^2$ */
else
  return (FALSE) /* q kein Quadrat */

5.1.2 Couveignes Methode

Die im vorletzten Abschnitt vorgestellte Methode zur Berechnung der Quadratwurzel scheitert in der Praxis daran, daß am Ende der Berechnung mit zu großen Zahlen gearbeitet werden muß.
J. M. Couveignes ([Cv94]) verbesserte diese Methode, so daß sie für die Faktorisierung kleinerer Zahlen verwendet werden konnte. Dieses Verfahren nutzt anstatt einer Primzahl modulo der die Wurzel gezogen wird, mehrere. Für die so berechneten Wurzeln δ_p gilt

$$\delta_p \ Mod\, p = \delta \ Mod\, p,$$

wobei Mod in diesem Falle bedeutet, daß die Koeffizienten von $\delta_p\, Mod\, p$ bzw. von $\delta\, Mod\, p$ in $\{0, \ldots, p-1\}$ liegen.
Wendet man auf die δ_p den jeweiligen Homomorphismus

$$\begin{array}{rcl} \psi_p : \mathbb{Z}/p\mathbb{Z}[\omega] & \rightarrow & \mathbb{Z}/p\mathbb{Z} \\ \omega & \mapsto & f_D \cdot m \bmod p \end{array}$$

an, so ergibt sich für die resultierenden Zahlen $x_p = \psi_p(\delta_p)$

$$x_p\, Mod\, p = \psi(\delta)\, Mod\, p. \tag{5.8}$$

Die gesuchte Zahl $\psi(\delta)$ ergibt sich dann durch Anwendung des Chinesischen Restsatzes ([Kn81, 4.3.2, Theorem C]) bei anschließender Reduktion modulo n. Auf

den ersten Blick müßte dabei aber wiederum mit Zahlen gerechnet werden, die in der gleichen Größenordnung wie die Wurzelkoeffizienten liegen. Dies kann dadurch umgangen werden, daß bei der Anwendung des Chinesischen Restsatzes (CRA) die Zwischenergebnisse schon modulo n reduziert werden.
Seien $m_i \in \mathbb{N}_{>1}$, alle m_i paarweise teilerfremd. Sei weiter

$$M = \prod_i m_i \tag{5.9}$$

$$M_i = M/m_i \tag{5.10}$$

$$a_i = 1/M_i \pmod{m_i} \tag{5.11}$$

dann liefert der Chinesische Restsatz angewandt auf das Kongruenzensystem

$$x \equiv x_i \pmod{m_i}$$

für $x_i \in \{0, \ldots, m_i - 1\}$, den Vertreter z der Restklasse modulo M, die die Kongruenzen erfüllt. Für diesen gilt

$$z = \sum_i a_i \cdot M_i \cdot x_i.$$

Für unser Problem benötigen wir den betragsmäßig kleinsten Vertreter x dieser Restklasse, da dieser genau $\psi(\delta)$ ist. x erhält man durch Rechnen modulo M. Beträgt der ganzzahlige Anteil von z dividiert durch M

$$q = \left\lfloor \frac{z}{M} \right\rceil,$$

so gilt für x

$$x = z - M \cdot q \tag{5.12}$$

$$x = z - M \cdot \left(\left\lfloor \sum_i a_i \cdot M_i \cdot x_i / M \right\rceil \right) \tag{5.13}$$

$$x = z - M \cdot \left(\left\lfloor \sum_i \frac{a_i \cdot x_i}{m_i} \right\rceil \right). \tag{5.14}$$

Will man x modulo n berechnen, so läßt sich dies natürlich dadurch erreichen, daß man die einzelnen Summanden bzw. Faktoren modulo n reduziert

$$\bar{x} = \bar{z} - \bar{M} \cdot \bar{q}.$$

Der Querstrich zeigt dabei an, daß modulo n gerechnet wird. Die komplette Berechnung von $\bar{z}$ kann modulo n reduziert durchgeführt werden

$$\bar{z} = \sum_i \bar{a}_i \cdot \bar{M}_i \cdot \bar{x}_i.$$

Ebenso läßt sich $\bar{M}$ modulo n bestimmen. Probleme bereitet lediglich der Quotient q. Dieser ergibt sich nach (5.14) zu

$$q = \left\lfloor \sum_i \frac{a_i \cdot x_i}{m_i} \right\rceil.$$

Dabei liegen die a_i, x_i und m_i jeweils in der Größenordnung von m_i. Der Bruch $\frac{a_i \cdot x_i}{m_i}$ beträgt höchstens m_i, so daß sich q nur durch Aufaddieren einiger Zahlen der gleichen Größenordnung wie m_i bestimmen läßt.
Ein Problem, das die gerade beschriebene Vorgehensweise aufwirft, habe ich bisher verschwiegen: Es gibt nämlich zwei verschiedene Wurzeln von γ: $\pm\delta$. Die Berechnung in den Körpern $\mathbb{Z}/p\mathbb{Z}[\omega]$ führt dann für jedes p ebenfalls zu zwei verschiedenen Wurzeln $\pm\delta_p$. Diese führen zu zwei möglichen Werten $\pm x_p$. Damit mit dem Chinesischen Restsatz ein brauchbares Ergebnis zusammengesetzt werden kann, muß garantiert werden, daß die jeweils ausgewählten δ_p der gleichen Wurzel δ entsprechen. Dies läßt sich im Falle eines ungeraden Körpergrades mit Hilfe der Norm gewährleisten. Dann gilt nämlich

$$N(-\alpha) = -N(\alpha)$$

für $\alpha \in \mathbb{Z}[\omega]$. Die Norm von δ entspricht natürlich der Wurzel der Norm von γ. Man kann sie sehr leicht berechnen, denn es gilt

$$\begin{aligned}
N(f_D \cdot a + b \cdot \omega) &= N(f_D \cdot (a + b \cdot \rho)) && \text{wegen Lemma 2.3} \\
&= N(f_D) \cdot N(a + b \cdot \rho) && \text{Multiplikativität der Norm} \\
&= {f_D}^D \cdot N_f(a,b)/f_D && \text{nach [BuLePo93, Kap. 12.6, Seite 89]} \\
&= {f_D}^{D-1} \cdot N_f(a,b).
\end{aligned}$$

Die Norm von γ beträgt also

$$N(\gamma) = N(g'(\omega))^2 \cdot {f_D}^{(D-1)\cdot|S|} \cdot \prod_{(a,b)\in S} N_f(a,b) \tag{5.15}$$

und die Norm von δ somit

$$N(\delta) = \pm N(g'(\omega)) \cdot {f_D}^{\frac{(D-1)\cdot|S|}{2}} \cdot \sqrt{\prod_{(a,b)\in S} N_f(a,b)}.$$

Die Wurzel aus dem Produkt läßt sich einfach anhand der in der Siebphase berechneten Exponetenvektoren ermitteln.
Um zu garantieren, daß alle Werte x_p der gewünschten Wurzel entsprechen, muß man beachten, daß gilt

$$N(\delta) \equiv N(\delta_p) \bmod p.$$

Man muß sich zuerst auf eine Wurzel δ festlegen, indem man sich z.B. für positive Norm entscheidet. Die Berechnung einer Wurzel δ_p erfolgt, wie in Abschnitt 5.1.1 beschrieben. Das Vorzeichen wird dann durch Vergleichen der Werte $N(\delta_p)\,Mod\,p$ und $N(-\delta_p)\,Mod\,p$.
Für gerade Körpergrade ist eine Unterscheidung der Wurzeln $\pm\delta$ mittels der Norm nicht möglich. Hier bleibt nur, alle Möglichkeiten durchzuprobieren. Nutzt man l verschiedene Moduln, so müssen im schlechtesten Fall 2^{l-1} Kombinationen getestet werden, was in der Praxis (siehe Beispiel 5.4) aufgrund der Vielzahl der Möglichkeiten nicht zu leisten ist. Dies ist der einzige Grund, warum im NFS nur Polynome mit ungeradem Grad benutzt werden.

Couveignes Algorithmus läßt sich analog zu obigem Algorithmus ([BuLePo93, Kapitel 9]) ebenfalls mit Primzahlpotenzen durchführen. Der Vorteil liegt darin, daß weniger Primzahlen benötigt werden, modulo derer das algebraische Quadrat berechnet werden muß. Unter der Verwendung von Couveignes Idee wird die Größe der Zahlen, mit denen gearbeitet werden muß, maximal so groß wie das Maximum der höchsten Primzahlpotenz und n.

Beispiel 5.4 (Laufzeit von Couveignes Methode)
Couveignes Methode benötigt auf einer 21 MIPS–Maschine zum Wurzelziehen für Z_{80} (siehe Beispiel 2.29 auf Seite 38) etwa 70 Tage. Dabei wurden 578 6–stellige träge Primzahlen in 1024–ter Potenz verwendet.

5.1.3 Vorhersage der Größe der Koeffizienten

Eine gewichtige Rolle für die Laufzeit von Couveignes Methode spielt die Anzahl der Dezimalstellen der Wurzelkoeffizienten, da abhängig davon die Anzahl der Moduln und die Höhe der Potenzen bestimmt werden müssen. Zu Beginn der Ausführung von Couveignes Algorithmus sind dies allerdings noch unbekannte Größen. Man könnte nun so vorgehen, daß man das Verfahren für einige Primzahlen in beliebiger Potenz durchführt. Liefert dies nicht das gewünschte Ergebnis, d.h. die Anzahl der Moduln war zu gering, so berechnet man für weitere Primzahlen die Wurzel, wobei man allerdings die zuvor erzielten Ergebnisse x_p wieder verwenden kann. Dennoch ist diese Vorgehensweise ineffizient, da der Chinesische Restsatzalgorithmus oftmals unnötig aufgerufen wird. Schöner wäre es, die Anzahl der benötigten Moduln vorher zu bestimmen. Nach [Cv94, Kapitel 2.2] gilt für den i–ten Wurzelkoeffizienten w_i von δ

$$\begin{aligned} |w_i| &\leq \left| \sum_{k=1}^{D} D^{\frac{1}{2}} \cdot ||g(x)||^{D-i} \cdot \delta^{(k)} / g'\left(\omega^{(k)}\right) \right| \\ &\leq D^{\frac{3}{2}} \cdot ||g(x)||^{D-i} \cdot \max_{k=1,\ldots,D} \left| \delta^{(k)} / g'\left(\omega^{(k)}\right) \right| \\ &=: wurzel_i \end{aligned} \tag{5.16}$$

wobei $||g(x)||$ als

$$||g(x)|| = \left(\sum_{i=0}^{D} {g_i}^2 \right)^{\frac{1}{2}}$$

definiert ist, und mit $\delta^{(k)}$ bzw. $\omega^{(k)}$ die k–te Konjugierte von δ bzw. ω bezeichnet wird.
Eine Abschätzung für die Dezimalstellenanzahl von w_i ergibt sich dann durch Logarithmieren der Gleichung (5.16) zur Basis 10. Der Wert

$$\max_{i=1,\ldots,D} \log_{10} |wurzel_i|$$

kann dann als obere Grenze für die Anzahl der Dezimalstellen der Wurzelkoeffizienten genommen werden.

5.2 Reduktion der Quadrate

Die Laufzeit des im Kapitel 5.1 beschriebenen Algorithmus hängt wesentlich von der Größe der Koeffizienten der Wurzel aus dem algebraischen Quadrat ab. Im folgenden möchte ich ein Verfahren vorstellen, das ausgehend von der Menge S von Paaren (a, b) ein algebraisches Quadrat konstruiert, dessen Wurzel Koeffizienten mit wesentlich weniger Dezimalstellen besitzt.

5.2.1 Umbildung der Quadrate

Bisher haben wir immer die Quadrate

$$x^2 = g'(f_D \cdot m)^2 \cdot {f_D}^{|S|} \cdot \prod_{(a,b)\in S} (a + bm) \tag{5.17}$$

auf der rationalen und

$$\gamma = g'(\omega)^2 \cdot \prod_{(a,b)\in S} (f_D \cdot a + b\omega)$$

auf der algebraischen Seite betrachtet. Setzt man aber

$$\gamma^{\pm} = \prod_{(a,b)\in S} (f_D \cdot a + b\omega)^{\lambda(a,b)}$$

mit $\lambda(a, b) \in \{-1, 1\}$, und

$$S^+ = \{(a, b) \in S \mid \lambda(a, b) = 1\}, \tag{5.18}$$

$$S^- = \{(a, b) \in S \mid \lambda(a, b) = -1\}, \tag{5.19}$$

so ist

$$S = S^+ \cup S^-. \tag{5.20}$$

Definiert man weiter

$$\gamma^+ = \prod_{(a,b)\in S^+} (f_D \cdot a + b\omega) \in \mathbb{Z}[\omega] \tag{5.21}$$

$$\gamma^- = \prod_{(a,b)\in S^-} (f_D \cdot a + b\omega) \in \mathbb{Z}[\omega], \tag{5.22}$$

so gilt $\gamma^- \neq 0$, da $(0, 0) \notin S$. Somit ist $\gamma^{\pm}$ wohldefiniert und es gilt

$$\gamma^{\pm} = \frac{\gamma^+}{\gamma^-} \in \mathbb{Q}(\omega) \tag{5.23}$$

und natürlich

$$\gamma = g'(\omega)^2 \cdot \gamma^+ \cdot \gamma^- \in \mathbb{Z}[\omega].$$

Allerdings handelt es sich bei $\gamma^{\pm}$ nicht mehr unbedingt um ein Element aus $\mathbb{Z}[\omega]$. Nach Bemerkung 1.9 von Seite 10 gibt es $z \in \mathbb{N}$ mit

$$\gamma^{\pm} \cdot z^2 \in \mathbb{Z}[\omega].$$

Für ein solches z gilt dann

$$\begin{aligned}
(x \cdot z)^2 &= x^2 \cdot z^2 \\
&\equiv \psi(\gamma) \cdot z^2 \\
&\equiv \psi\left(\gamma \cdot z^2\right) \\
&\equiv \psi\left(g'(\omega)^2 \cdot \gamma^+ \cdot \gamma^- \cdot z^2\right) \\
&\equiv \psi\left(g'(\omega)^2 \cdot \frac{\gamma^+ \cdot z^2}{\gamma^-} \cdot \left(\gamma^-\right)^2\right) \\
&\equiv \psi\left(g'(\omega)^2 \cdot \gamma^\pm \cdot z^2 \cdot \left(\gamma^-\right)^2\right) \\
&\equiv \psi\left(g'(\omega)^2 \cdot \underbrace{\gamma^\pm \cdot z^2}_{\in \mathbb{Z}[\omega]}\right) \cdot \psi\left(\gamma^-\right)^2 \pmod{n}
\end{aligned}$$

Unter der Einschränkung

$$\psi\left(\gamma^-\right) \in \mathbb{Z}/_{n\mathbb{Z}}{}^*, \tag{5.24}$$

d.h. $\psi(\gamma^-)$ ist modulo n invertierbar, ist die Division beider Seiten der Kongruenz durch $\psi(\gamma^-)^2$ möglich. Man erhält dann die quadratische Kongruenz

$$\left(x \cdot z \cdot \psi\left(\gamma^-\right)^{-1}\right)^2 \equiv \psi\left(g'(\omega)^2 \cdot \gamma^\pm \cdot z^2\right) \pmod{n}. \tag{5.25}$$

Da auf der algebraischen Seite das Ausgangsquadrat γ zum einen nur durch ein Quadrat dividiert ($(\gamma^-)^2$) und zum anderen mit einem Quadrat multipliziert (z^2) wurde, ist

$$\gamma^\pm \cdot z^2$$

ein Quadrat in $\mathbb{Z}[\omega]$ und

$$g'(\omega)^2 \cdot \gamma^\pm \cdot z^2$$

sogar ein Quadrat in $\mathbb{Z}[\omega]$, dessen Wurzeln in $\mathbb{Z}[\omega]$ liegen (siehe Seite 36).
Die Wurzel aus der linken Seite der Kongruenz läßt sich sehr leicht berechnen, denn es gilt

$$\begin{aligned}
\left(x \cdot z \cdot \psi\left(\gamma^-\right)^{-1}\right)^2 &\equiv \frac{g'(f_D \cdot m)^2 \cdot f_D{}^{|S|} \cdot \prod\limits_{(a,b)\in S}(a+bm) \cdot z^2}{\psi(\gamma^-)^2} && \text{nach (5.17)} \\
&\equiv \frac{z^2 \cdot g'(f_D \cdot m)^2 \cdot f_D{}^{|S|} \cdot \prod\limits_{(a,b)\in S}(a+bm)}{f_D{}^{2\cdot|S^-|} \cdot \left(\prod\limits_{(a,b)\in S^-}(a+bm)\right)^2} && \text{nach (5.22) u. Lemma 2.3} \\
&\equiv f_D{}^{|S|-2\cdot|S^-|} \cdot z^2 \cdot \frac{\prod\limits_{(a,b)\in S^-}(a+bm) \cdot \prod\limits_{(a,b)\in S^+}(a+bm)}{\left(\prod\limits_{(a,b)\in S^-}(a+bm)\right)^2} && \text{nach (5.20)} \\
&\equiv f_D{}^{|S|-2\cdot|S^-|} \cdot z^2 \cdot \frac{\prod\limits_{(a,b)\in S^+}(a+bm)}{\prod\limits_{(a,b)\in S^-}(a+bm)} \pmod{n},
\end{aligned}$$

also

$$x \cdot z \cdot \psi\left(\gamma^{-}\right)^{-1} \equiv f_D{}^{|S|/2-|S^-|} \cdot z \cdot \frac{\sqrt{\prod\limits_{(a,b)\in S^+}(a+bm)}}{\sqrt{\prod\limits_{(a,b)\in S^-}(a+bm)}} \pmod{n} \tag{5.26}$$

$f_D{}^{\frac{|S|}{2}-|S^-|}$ läßt sich mit Algorithmus FASTEXPO von Seite 161 schnell berechnen. Ist $\frac{|S|}{2}-|S^-|$ negativ, so muß man mit FASTEXPO $f_D{}^{-(\frac{|S|}{2}-|S^-|)}$ berechnen und das Ergebnis modulo n invertieren, was wegen der Einschränkung (5.24) möglich ist.
Da die Primfaktorzerlegungen von $a+bm$ in den entsprechenden Exponentenvektoren abgelegt ist, läßt sich daraus leicht die Primfaktorzerlegung des Bruches berechnen:

$$\prod_{(a,b)\in S^+}(a+bm) = \prod_{p_i\in FB_R} p_i^{\sum\limits_{(a,b)\in S^+} e_i^{(a,b)}} \tag{5.27}$$

$$\prod_{(a,b)\in S^-}(a+bm) = \prod_{p_i\in FB_R} p_i^{\sum\limits_{(a,b)\in S^-} e_i^{(a,b)}} \tag{5.28}$$

$$\begin{aligned}\frac{\prod\limits_{(a,b)\in S^+}(a+bm)}{\prod\limits_{(a,b)\in S^-}(a+bm)} &= \prod_{p_i\in FB_R} p_i^{\sum\limits_{(a,b)\in S^+} e_i^{(a,b)} - \sum\limits_{(a,b))\in S^-} e_i^{(a,b)}} \\ &= \prod_{p_i\in FB_R} p_i^{\sum\limits_{(a,b)\in S} \lambda(a,b)\cdot e_i^{(a,b)}} \qquad (5.29)\end{aligned}$$

Da obiger Bruch ein Quadrat ist, sind alle Exponenten gerade, somit erhält man die Wurzel des Bruches durch Berechnung von

$$\prod_{p_i\in FB_R} p_i^{\frac{1}{2}\cdot\sum\limits_{(a,b)\in S}\lambda(a,b)\cdot e_i^{(a,b)}} \pmod{n}.$$

Primzahlen mit negativem Exponenten müssen zuerst mit dem Betrag des Exponenten potenziert und das Ergebnis modulo n invertiert werden. Um Invertierungen einzusparen, kann man auch zuerst das Produkt aller Primzahlen mit negativen Exponenten in entsprechender positiver Potenz bilden, und anschließend das Inverse berechnen.

Der Vorteil der gerade beschriebenen Methode liegt darin, daß man durch geschicktes Wählen der $\lambda(a,b)$, also der Teilmengen S^+ und S^- von S, die Größe des algebraischen Quadrates, aus dem schließlich die Wurzel gezogen werden muß, stark reduzieren kann. Zur Vorhersage der Stellenanzahl der Wurzelkoeffizienten kann man analog zu Kapitel 5.1.3 vorgehen.
Zur Berechnung der Wurzel aus dem neuen Quadrat kann man die Methode von Couveignes verwenden. Ebenso ist der Einsatz von höheren Potenzen möglich.

5.2.2 Die Teilmengen S^+ und S^-

Der algebraische Teil des Exponentenvektors $\vec{e}_S$ (siehe Bemerkung 2.24)

$$f_1, \ldots, f_{B2}$$

ist ein gutes Maß für die Größe des algebraischen Produktes. Je größer die Einträge umso größer sind auch die Quadrate. Allerdings spielen auch die algebraischen Large Primes eine wichtige Rolle. Deshalb erweitern wir an dieser Stelle den algebraischen Teil des Exponentenvektors $\vec{e}(a, b)$ auf $\tilde{B}2$ Einträge und nehmen die vorkommenden algebraischen Large Primes in die Faktorbasis auf

$$\vec{e}_{alg} = (f_1, \ldots, f_{\tilde{B}_2}).$$

Will man das algebraische Produkt reduzieren, so ist ein mögliches Kriterium für die Reduziertheit die Summe der Beträge der Komponenten von $\vec{e}_{alg}$. Je geringer dieser Wert ist, umso besser ist das Quadrat reduziert, wenn man einmal von Einheiten absieht.
Es sei im folgenden $\vec{e}$ die Summe der Exponentenvektoren der Paare aus S^+ abzüglich der Summe der Exponentenvektoren aus S^-. Weiter liefere sum angewandt auf einen Exponentenvektor die Summe der Beträge der Einträge aus dem algebraischen Teil dieses Vektors.
Das in Algorithmus 5.5 präsentierte Verfahren zur Aufteilung von S in die disjunkten Teilmengen S^+ und S^- liefert in der Praxis stark reduzierte Quadrate. Dabei geht man von der mit S initialisierten Menge S^+ und $S^- = \emptyset$ aus. Es werden nun nacheinander Paare (a, b) aus S^+ entfernt und zu S^- hinzugefügt, und zwar immer dann, wenn man dadurch jeweils einen kleineren Wert $sum(\vec{e})$ erhält. Es ist durch diese Vorgehensweise möglich, daß man weitere Verbesserungen bekommt, wenn man nach der Bearbeitung der gesamten Menge S^+ wieder einige Paare aus S^- herauslöst und S^+ zuordnet. Aus diesem Grund müssen S^+ und S^- nacheinander solange betrachtet werden, bis ein Durchlauf durch beide Mengen keine Verringerung des Wertes $sum(\vec{e})$ mehr bewirkt. Diese Vorgehensweise ist endlich, da in jedem Lauf durch S^+ und S^- der Wert $sum(\vec{e})$, der ja in $\mathbb{N}_0$ liegt, höchstens verringert, niemals aber erhöht wird. Somit bricht das Verfahren spätestens bei $sum(\vec{e}) = 0$ ab.

Algorithmus 5.5 (Reduktion des Exponentenvektors)

REDEXPOVEC

EINGABE: S
AUSGABE: S^+, S^-, $\vec{e}$

(1) $S^+ = S$; $S^- = \emptyset$ /* Initialisierung */
(2) $\vec{e} := (0, \ldots, 0)$ /* Initialisierung */

Fortsetzung auf Seite 120

```
foreach ((a,b) ∈ S) do
   e⃗ = e⃗ + e⃗(a,b)                          /* e⃗ = ∑ e⃗(a,b) */
od
repeat
   vglsum = sum(e⃗)                          /* Summe der Eintragsbeträge zu
                                               Beginn der Schleife */
   foreach ((a,b) ∈ S⁺) do                  /* fd·a + bω bisher multipliziert */
      if (sum(e⃗ − 2·e⃗(a,b))                 /* Summe der Eintragsbeträge
            < sum(e⃗)) then                     vergleichen */
         S⁺ = S⁺ − {(a,b)}
         S⁻ = S⁻ ∪ {(a,b)}                  /* fd·a + bω dividieren */
         e⃗ := e⃗ − 2·e⃗(a,b)                  /* Exponentenvektor korrigieren */
      fi
   od
   foreach ((a,b) ∈ S⁻) do                  /* fd·a + bω bisher dividiert */
      if (sum(e⃗ + 2·e⃗(a,b))                 /* Summe der Eintragsbeträge
            < sum(e⃗)) then                     vergleichen */
         S⁻ = S⁻ − {(a,b)}
         S⁺ = S⁺ ∪ {(a,b)}                  /* fd·a + bω multiplizieren */
         e⃗ := e⃗ + 2·e⃗(a,b)                  /* Exponentenvektor korrigieren */
      fi
   od
until (vglsum = sum(e⃗))                     /* keine Änderungen mehr */
return (S⁺, S⁻, e⃗)
```

5.2.3 Das Korrekturquadrat

Nach (5.23) gilt $\gamma^{\pm} \in \mathbb{Q}(\omega)$. Um wieder ein Element aus $\mathbb{Z}[\omega]$ zu erhalten, ohne dabei die Quadrateigenschaft zu verlieren, suchen wir ein $z^2 \in \mathbb{Z}$, so daß

$$z^2 \cdot \gamma^{\pm} \in \mathbb{Z}_w.$$

Als notwendige Voraussetzung dafür, daß eine algebraische Zahl in $\mathbb{Z}[\omega]$ liegt, gilt, daß die Norm ganzzahlig ist.
Nach (5.15) folgt für die Norm von $\gamma^{\pm}$

$$N(\gamma^{\pm}) = {f_D}^{(D-1)(|S^+|-|S^-|)} \cdot \frac{\prod\limits_{(a,b)\in S^+} N_f(a,b)}{\prod\limits_{(a,b)\in S^-} N_f(a,b)} \quad .$$

O.B.d.A können wir davon ausgehen, daß $|S^+| \geq |S^-|$ ist. Dann muß zur Erfüllung der Normbedingung nur gewährleistet werden, daß der Bruch aus $\mathbb{Z}$ ist. Dies läßt sich aber einfach an dem Vektor $\vec{e}$ erkennen. Sind alle Einträge nicht negativ, so ist der Bruch in $\mathbb{Z}$.
Jede Stelle des Vektors $\vec{e}$ korrespondiert zu einem Paar (p, cp) aus R und somit zu einem Primideal in A (siehe (2.16) und (2.26)). Ebenso korrespondieren diese Paare

mit Ausnahme der Paare, deren erste Komponente $[\mathcal{O}_\omega : \mathbb{Z}[\omega]]$ teilt, aufgrund der Tatsache

$$g(f_D \cdot cp) \equiv f(f_D \cdot cp/f_D) \equiv 0 \pmod p$$

zu den Paaren $(p, f_d \cdot cp)$. Diese entsprechen ihrerseits wiederum den Primidealen primer Norm von $\mathcal{O}_\omega$ (siehe Satz 1.27). Somit gibt der Vektor $\vec{e}$ bis auf die den Indexteilern entsprechenden Einträgen, die Zerlegung des gebrochenen Ideals

$$\gamma^{\pm} \cdot \mathcal{O}_\omega$$

in Primideale primer Norm wieder.
Für Primideale, deren Norm von einem Indexteiler geteilt wird, gilt nicht mehr unbedingt, daß der ihnen entsprechende Eintrag im Exponentenvektor, dem Exponeneten entspricht, mit dem sie in $\gamma^{\pm}$ vorkommen. Ein Beispiel dafür, was durch Ideale passieren kann, die einem Paar (p, cp) mit $p | [\mathcal{O}_\omega : \mathbb{Z}[\omega]]$ entsprechen, ist

Beispiel 5.6 ($\pi_{p,cp}$, das kein Primideal ist)
Nach [Ch93, Proposition 6.2.1] ist $\pi_{p,cp}$ ein Ideal von möglicherweise höherer p-Potenz und nicht mehr unbedingt ein Primideal. Sei $\pi_{p,f_D \cdot cp}$ nun ein Ideal der Norm p^3, $p \not| f_D$ das in die beiden Primideale $\mathcal{P}_1$ mit der Norm p und $\mathcal{P}_2$ mit der Norm p^2 zerfällt. Findet man z.B. zwei gute Paare (a_i, b_i), $i = 1, 2$, für die gilt

$$\begin{array}{rcl} \mathcal{P}_1 \cdot {\mathcal{P}_2}^4 & | & (f_D \cdot a_1 + b_1 \cdot \omega)\mathcal{O}_\omega \text{ und} \\ {\mathcal{P}_1}^5 \cdot {\mathcal{P}_2}^2 & | & (f_D \cdot a_2 + b_2 \cdot \omega)\mathcal{O}_\omega, \end{array}$$

so würden in beiden Exponentenvektoren, an der $\pi_{p,f_D \cdot cp}$ entsprechenden Stelle 9 stehen, da $\pi_{p,cp}$ in der Siebphase als Primideal der Norm p angesehen wird. Dividiert man $(f_D \cdot a_1 + b_1 \cdot \omega)\mathcal{O}_\omega$ durch $(f_D \cdot a_2 + b_2 \cdot \omega)\mathcal{O}_\omega$, so bleibt im Zähler noch ${\mathcal{P}_2}^2$ und im Nenner ${\mathcal{P}_1}^4$ stehen, obwohl sich p in der Norm wegkürzt. Das Primideal $\mathcal{P}_1$ sorgt iin diesem Fall dafür, daß $\gamma^{\pm}$ nicht in $\mathbb{Z}[\omega]$ liegt. □

Zur Behebung des durch Primideale mit negativen Exponenten verursachten Defekts kann Lemma 1.24 heangezogen werden. In der zweiten Behauptung heißt es dort

$$N(\mathcal{I}) \in \mathcal{I},$$

was für unseren Fall bedeutet

$$\pi_{p,f_D \cdot cp} \mid p.$$

Hieraus folgt , daß die Zahl $\tilde{z}^2$ mit

$$\tilde{z} := \prod_{\substack{(p,cp) \in R \\ f_{(p,cp)} < 0}} p^{\left|\frac{f_{(p,cp)}}{2}\right|}$$

den von Primidealen $\pi_{p,f_D \cdot cp}$ mit $p \not| [\mathcal{O}_\omega : \mathbb{Z}[\omega]]$ begangenen Fehler behebt.
Um den von den Indexteilern verursachten Nenner durch Multiplikation verschwinden lassen zu können, müssen zuerst einmal die Indexteiler bestimmt werden. Dabei können wir uns auf die Primzahlen zurückziehen, die die Norm eines Elementes der algebraischen Faktorbasis oder einer algebraischen Large Prime teilen.

Nach Definition 1.26 und Lemma 1.16 gilt, daß das Quadrat jeden Indexteilers p in $N(g'(w))$ aufgehen muß, also

$$p^2 \mid |N(g'(w))| \tag{5.30}$$

gilt. Da für uns nur spezielle, d.h. kleine Indexteiler interessant sind, läßt sich durch Überprüfen der Bedingung (5.30) für alle p aus den Paaren (p, cp) der algebraischen Faktorbasis und allen Normen der verwendeten Large Prime Ideale ein Produkt z_{ind} von Primzahlen berechnen, das von allen in Frage kommenden Indexteilern geteilt wird.
Kennt man nun alle beteiligten Indexteiler, so läßt sich dennoch nur mit erheblichem zusätzlichen Aufwand berechen, in welche Primideale die entsprechenden Ideale $\pi_{p,cp}$ zerfallen, und mit welchem Exponenten diese Primideale im Nenner von $\gamma^{\pm}$ vorkommen. Aber die Wahrscheinlichkeit, daß durch die Kontruktion von $\gamma^{\pm}$ diese Exponenten weitgehend verschwinden, ist sehr hoch. Deshalb empfiehlt es sich, einen geraden Exponenten e_{ind} zu wählen, mit dem das Produkt z_{ind} potenziert wird

$$z = \tilde{z}^2 \cdot z_{ind}^{e_{ind}}. \tag{5.31}$$

Wählt man diesen Exponenten zu nierig, so liegt also kein Quadrat in $\mathbb{Z}[\omega]$ vor, wohl aber eins aus $\mathbb{Q}(\omega)$ und damit auch in $\mathbb{Z}/p\mathbb{Z}[\omega]$. Das oben beschriebene Verfahren kann also weitergeführt werden, allerdings liefern die beiden im Schritt (4) des NFS–Algorithmus 2.4 gebildeten größten gemeinsamen Teiler

$$\mathrm{ggT}(x+y, n) \text{ und } \mathrm{ggT}(x-y, n)$$

jeweils 1, da die quadratische Kongruenz nicht mehr korrekt ist. In diesem Fall muß die Berechnung für einen höheren Wert e_{ind} wiederholt werden. In der Praxis hat sich $e_{ind} = 10$ als völlig ausreichend erwiesen. In keiner von mir durchgeführten Berechnung wurde eine höhere Potenz von z_{ind} benötigt.
Für Primideale, deren korrespondierendes Paar in der ersten Komponente einen Teiler von f_D besitzen, gilt das analoge: Man kann anhand des Exponentenvektors nicht erkennen, in welcher Potenz diese Primideale vorkommen. In diesen Fällen kommt noch erschwerend hinzu, daß die Anzahl der Linearfaktoren von $f(x) \pmod p$ und $g(x) \pmod p$ sich unterscheiden können, so daß man an $\vec{e}$ schon gar nicht mehr das Zerfallsverhalten ablesen kann. Dennoch können diese Indexteiler ebenso wie die anderen Indexteiler behandelt werden.

Durch das oben beschriebene Verfahren läßt sich die Stellenanzahl der Koeffizienten der algebraischen Wurzel stark reduzieren. Dadurch werden weniger Primzahlen benötigt, modulo der die Wurzel berechnet werden muß. Tabelle 5.1 zeigt für einige verschiedene faktorisierte Zahlen, die Aufteilung der Paare (a, b) in die Teilmengen S^+ und S^- sowie die maximale Stellenanzahl der Wurzelkoeffizienten. Dabei wurde immer die erste Menge S verwendet, die zu einer nichttrivialen Faktorisierung von n führte.

Tabelle 5.1: maximale Stellenanzahl der Wurzelkoeffizienten

n	ohne Reduktion		mit Reduktion			Verhältnis
	$\lvert S\rvert$	Stellenanzahl s_1	$\lvert S^+\rvert$	$\lvert S^-\rvert$	Stellenanzahl s_2	$^{s_1}/_{s_2}$
Z_{10}	84	203	44	40	78	2.6
Z_{20}	372	979	187	185	184	5.3
Z_{30}	1958	9120	1005	953	1179	7.7
Z_{40}	4306	27352	2201	2105	3663	7.5
Z_{50}	11094	84597	5651	5443	10372	8.2
Z_{60}	45580	410907	23208	22372	37717	10.9
Z_{80}	338064	2959333	170107	167957	184698	16.0

5.3 Quadratberechnung in der Praxis

Analysiert man, wie sich die Laufzeit auf die einzelnen Schritte der Quadratwurzelberechnung verteilt, so kann man erkennen, daß fast die gesamte Zeit für die Berechnung des Produkts $\gamma^{\pm}$ modulo der Primzahlpotenzen benötigt wird (siehe Tabelle 5.2). Realisiert man diesen Schritt sukzessive, d.h. liest man ein Paar (a, b) ein und multipliziert $f_D \cdot a + b\omega$ auf das entsprechende Teilprodukt γ^- oder γ^+ in $\mathbb{Z}/p^l\mathbb{Z}[\omega]$, so werden bei pz verschiedenen Potenzen, etwa

$$|S| \cdot pz$$

Multiplikationen modulo benötigt.
Bei der Faktorisierung der Zahl Z_{55} (siehe Kapitel 7.6) wurden 15 Primzahlen der Größenordnung 10^5 in 256–ter Potenz benötigt, um die Wurzel berechnen zu können. Tabelle 5.2 zeigt die Verteilung der Laufzeit auf die einzelnen Schritte am Beipiel Z_{55}.

Tabelle 5.2: Verteilung der Laufzeit beim Wurzelziehen auf die einzelnen Schritte

Teilschritt	in Sekunden	in Prozent
REDEXPOVEC	160.47	0.7
Mult. in $\mathbb{Z}/p^l\mathbb{Z}[\omega]$	21753.6	98.0
Wurzel modulo p	16.77	0.08
CRA	89.08	0.4
total	22191.2	100

Eine starke Reduktion der Anzahl von durchzuführenden Multiplikationen läßt sich dadurch erreichen, daß man einige Multiplikationen in $\mathbb{Z}[\omega]$ durchführt, das Resultat modulo der Primzahlpotenzen reduziert und dann erst zu den entsprechenden Teilprodukten multipliziert. Sei t die Anzahl der Faktoren, die in $\mathbb{Z}[\omega]$ jeweis auf-multipliziert werden, so reduziert sich die Anzahl der Multiplikationen auf

$$|S| \text{ Multiplikationen in } \mathbb{Z}[\omega] \text{ und } \frac{|S|}{t} \cdot pz \text{ Multiplikationen in } {}^{\mathbb{Z}}\!/_{p^l\mathbb{Z}}[\omega].$$

Hierbei ist zu beachten, daß mit steigender Zahl von Multiplikationen ohne Reduktion, die Stellenanzahl der Koeffizienten der Zwischenergebnisse stark ansteigt und so die Multiplikationen verlangsamt. Der beste Weg ist hier, die Multiplikationen in $\mathbb{Z}[\omega]$ baumartig durchzuführen, so daß immer in etwa gleich große Zahlen miteinander multipliziert werden. Tabelle 5.3 zeigt die Verteilung der Laufzeit beim Wurzelziehen, wenn man $t = 256$ wählt.

Tabelle 5.3: Verteilung der Laufeit beim Wurzelziehen unter Verwendung eines Baumes

Teilschritt	in Sekunden	in Prozent
REDEXPOVEC	160.47	5.3
Mult. in $\mathbb{Z}/p^l\mathbb{Z}[\omega]$	2435.38	80.60
Mult. in $\mathbb{Z}[\omega]$	148.44	4.9
Wurzel modulo p	16.77	0.5
CRA	89.08	2.9
total	3021.42	100

Man sieht, daß sich die Laufzeit durch die Verwendung der gerade beschriebenen Variante in diesem Beispiel auf etwa ein Siebtel reduzieren läßt.
Vergrößert man weiter die Anzahl der Multiplikationen, die ohne Reduktion durchgeführt werden, so läßt sich die Gesamtlaufzeit noch weiter verbessern, bis zu dem Zeitpunkt, an dem die Zahlen so groß werden, daß die Multiplikationen in $\mathbb{Z}[\omega]$ und die anschließenden Reduktionen modulo der Primzahlpotenzen gegenüber den Multiplikationen in $\mathbb{Z}/p^l\mathbb{Z}[\omega]$ zu teuer werden. Die folgende Tabelle 5.4 zeigt am Beipiel Z_{55}, wie sich die Laufzeit für das Bilden des Produktes $z^2 \cdot \gamma$ verändert, wenn man die Parameter t, Anzahl der Primzahlen und Potenz der Moduln variiert.
Die beste Laufzeit erhält man in diesem Beispiel bei der Verwendung von 8 Moduln in 512–ter Potenz (also etwa 20480 Dezimalstellen) bei 1024 Multiplikationen in $\mathbb{Z}[\omega]$ ohne zu reduzieren. Als Zwischenergebnis können dabei algebraische Zahlen mit Koeffizienten von etwa 17500 Dezimalstellen auftreten.

5.4 Verteilte Quadratberechnung

Wie wir im letzten Abschnitt gesehen haben, geht der größte Teil der Laufzeit in die Berechnung des Produktes

$$g'(\omega)^2 \cdot \prod_{(a,b)\in S} (f_D \cdot a + b\omega)$$

Tabelle 5.4: Änderung des Laufzeitverhaltens bei verschiedenen Parametern in Sekunden

Anzahl Moduln	Exponent	Multiplikationen ohne Reduktion				
		256	512	1024	2048	4096
4	10	3542	3468	3132	3103	3136
8	9	3371	3030	2977	3054	3193
16	8	3222	3098	3010	3067	3242
32	7	3257	3176	3153	3240	3307

in $\mathbb{Z}/p^l\mathbb{Z}[\omega]$. Bei großen zu faktorisierenden Zahlen wird zuviel Zeit dazu benötigt, als daß man die Berechnung auf einem Rechner durchführen möchte. Deshalb bietet sich eine verteilte Bearbeitung des Problems an.
Eine Möglichkeit, die Aufgaben im Client–Server Modell (siehe Kapitel 3.6) zu verteilen, besteht im Aufteilen der Menge der Primzahlmoduln auf die Clients. Dort wird jeweils das algebraische Quadrat in den entsprechenden Ringen gebildet, die Wurzeln gezogen und das Ergebnis geliftet. Schließlich werden noch die Homomorphismen ψ_p angewandt und die resultierende Zahl zum Server gesendet. Dieser setzt die Ergebnisse der Clients mit dem Chinesischen Restsatz zusammen und bestimmt die Teiler von n.
Diese Art der Aufgabenverteilung hat in der Praxis einen entscheidenden Nachteil. Auf jedem Client müssen dann nämlich alle Relationen, die bei der Quadratbildung eine Rolle spielen verfügbar sein. Bei der Faktorisierung von Z_{80} wären dies zum Beispiel etwa 170–Megabyte.
Aus diesem Grunde empfiehlt es sich, auf dem Server einige Vorberechnungen durchzuführen. Zuerst werden dort die beiden Mengen S^+ und S^- bestimmt, und dann Teilprodukte in $\mathbb{Z}[\omega]$ unter Ausnutzung einer Baumstruktur berechnet. Diese Zahlen werden dann an alle Clients gesendet, wo sie modulo der Primzahlpotenzen weiterverarbeitet werden. Die weitere Vorgehensweise verläuft schließlich analog zu obigem Verfahren.

5.5 Zufälligkeit der Lösungen der quadratischen Kongruenz

Wie schon in der Bemerkung 2.5 auf Seite 18 erwähnt, konnte bisher noch nicht bewiesen werden, daß die im *Number Field Sieve* konstruierten Lösungen der quadratischen Kongruenz

$$(2.1) \qquad x^2 \equiv y^2 \pmod{n}$$

sich wie zufällige Lösungen verhalten. Um das Verhalten der Lösungen in der Praxis zu untersuchen, habe ich für die Zahlen Z_{10}, Z_{20}, Z_{30}, Z_{40}, Z_{50} (siehe Kapitel 7) die ersten 10 vom *Number Field Sieve* erzeugten Lösungen der quadratischen Kongruenz betrachtet. Das besondere an diesen fünf Zahlen ist, daß sie aus genau zwei Primzahlen zusammengesetzt sind. Nach den Betrachtungen aus Bemerkung 2.5 müßten etwa die Hälfte der Lösungen zu nichttrivialen Teilern führen. Die Tabelle 5.5 zeigt in der zweiten Spalte, wieviele Lösungen nichttriviale Teiler ergeben. Die dritte Spalte gibt die Anzahl der Lösungen wider, die nur triviale Teiler lieferten also zum Faktorisiseren ungeeignet sind. Die letzte Spalte zeigt schließlich, die wievielte Lösung zum ersten Mal einen nichttrivialen Teiler hervorbrachte.

Tabelle 5.5: Zufälligkeit der Lösungen

faktorisierte Zahl	Anzahl Lsg. mit nichttrivialer Teiler	Anzahl Lsg. mit trivialer Teiler	erste Lsg. mit nichttrivialem Teiler
Z_{10}[1]	3	7	1
Z_{20}	6	4	3
Z_{30}	5	5	1
Z_{40}	4	6	1
Z_{50}	5	5	3

Diese Tabelle bestätigt, daß sich die durch das *Number Field Sieve* generierten Lösungen wie zufällige Lösungen verhalten. Eine weitere Bekräftigung dieser Vermutung kann man davon ableiten, daß in allen von mir durchgeführten Faktorisierungen mit dem NFS nie mehr als drei Lösungen benötigt wurden, um einen nichttrivialen Faktor zu ermitteln.

[1]nimmt man die ersten 25 Lösungen, so stellt man ein Verhältnis 12 : 13 fest

Kapitel 6

Wahl der Parameter

In diesem Kapitel möchte ich darauf eingehen, welche Auswirkungen die einzelnen Parameter auf die Laufzeit der Siebphase und die nachfolgenden Schritte des NFS haben. Außerdem werde ich eine Tabelle angeben, in der in Abhängigkeit von der zu faktorisierenden Zahl Parameter aufgelistet sind, mit denen im allgemeinen die Faktorisierung effizient durchgeführt werden kann. Dabei werde ich voraussetzen, daß mit den im Kapitel 4 beschriebenen Vorgehensweisen das Polynom und dessen Wurzel modulo n berechnet wurden.

Die Siebphase des NFS–Algorithmus wird von vielen Parametern beeinflußt. In diesem Zusammenhang sind vor allem die Faktorbasen und die Large Prime Grenzen, aber auch die Größe des Siebintervalls und der Korrekturfaktor KORRFAK zu nennen. In den weiteren Betrachtungen werde ich davon ausgehen, daß die zu faktorisierende Zahl so groß ist, daß die Quadruple Large Prime Variante angewandt wird.

Die Größe der Faktorbasen

Gehen wir einmal davon aus, wir hätten feste Grenzen für die Large Primes vorgegeben. Wie soll man dann die Faktorbasisgrößen wählen? Nimmt man sie zu klein, so muß sehr lange gesiebt werden, um genügend viele Relationen zu finden. Der Siebvorgang wird dann ineffizient. Zu kleine Faktorbasen haben desweitern den Nachteil, daß sie bei der Verwendung von vier Large Primes zu riesigen Zykeln führen (siehe Seite 83). Dadurch wird zum einen die Matrix wesentlich dichter, was zu verlängerten Laufzeiten beim Lösen des linearen Gleichungssystems führt, zum anderen steigt die Anzahl der Paare der Menge S, die zur quadratischen Kongruenz führt, ebenfalls stark an. Dies wirkt sich negativ auf die Laufzeit der Berechnung der Wurzel aus dem algebraischen Quadrat aus.
Wählt man hingegen zu große Faktorbasen, so wirkt sich das zuerst einmal negativ auf die Laufzeit der Relationensuche aus. Einerseits muß für mehr Faktorbasiselemente gesiebt werden, was die Berechnung einer Relation natürlich verteuert und andererseits kann die in Kapitel 3.4 beschriebene Zykelexplosion nicht ausgenutzt werden, weil man schon bevor diese Explosion eintritt, genügend Relationen generiert hat. Weiterhin führen zu große Faktorbasen zu Matrizen mit größerer Dimension, was die Laufzeit für das Lösen des linearen Gleichungssystem negativ beeinflußt.

Die Large Prime Grenzen

Je größer man die Large Prime Grenzen wählt, umso mehr Relationen werden in der Relationensuche erzeugt und umso mehr steigt die Wahrscheinlichkeit einen Zykel zu finden. Allerdings nimmt die Wahscheinlichkeit, daß eine Large Prime in mehr als einer Relation vorkommt, mit ihrer Größe ab. Somit sind große Large Primes in bezug auf die Verwendbarkeit in Zykeln weniger effektiv. Durch übersteigerte Large Prime Grenzen erhöht sich aber auch die Laufzeit der Siebphase dadurch, daß die Zahl der Überlebenden des LOGSIEB–Algorithmus enorm ansteigt und somit auch die erlaufzeitintensiven Schritte Probedivision, Primzahltest und Zerlegung des Large Prime Produkts sehr oft durchgeführt werden müssen.
Wählt man dagegen die Large Prime Grenze zu klein, so muß, wie bei einer kleinen Faktorbasis, möglicherweise zu lange gesiebt werden bis genügend viele Relationen erzeugt wurden.

Die Siebgrenzen

Bei der Betrachtung der Siebarraygrößen muß man natürlich unterscheiden, ob das normale NFS oder Lattice Sieve durchgeführt wird. In beiden Fällen wirkt sich die Größe des Siebarrays nur auf die Laufzeit der Siebphase aus.
Im NFS Algorithmus stellt sich insbesondere die Frage, wie groß die Siebgrenzen A_{min} und A_{max} im Verhältnis zu der Grenze B_{max} gewählt werden sollen.
Betrachtet man die rationalen Werte $|a + bm|$ und die algebraischen $|N_f(a, b)|$, die zerlegt werden müssen, so sieht man, daß im rationalen Fall, a die Größenordnung von $|a + bm|$ in der Praxis überhaupt nicht beeinflüst, da $m >> |a|$ ist. Die Koeffizienten des Polynoms $f(x)$ sind im wesentlichen so groß wie m, so daß $|N_f(a, b)| = |f(-a/b) \cdot (-b)^D|$ in der Größenordnung von $\max(|a|, b)^D \cdot m$ liegen. Während die Erhöhung der zu untersuchenden Werte a um eine Zehnerpotenz im rationalen Fall keinen Unterschied in der Größe der zu zerlegenden Werte macht, nimmt die algebraische Norm um D Dezimalstellen zu. Deshalb ist es im Normalfall sinnvoll, das Siebrechteck AB so zu wählen, daß

$$-A_{min} = A_{max} = B_{max}.$$

In der Praxis kann man mit kleinen Siebarraygrenzen starten. Sollten nicht genügend viele Relationen gefunden worden sein, so siebt man einfach in einem noch nicht betrachteten Bereich weiter. Durch diese Vorgehensweise kann man verhindern, daß man im Bereich betragsmäßig sehr großer Werte a bzw. b, wo erfahrungsgemäß sehr wenige gute Paare liegen, zu lange siebt, und Bereiche, in denen sehr viele gute Paare zu finden sind, möglicherweise gar nicht mehr betrachten muß.
Verwendet man zur Relationengenerierung das Lattice Sieve, so muß man unter Übertragung der Siebgrenzen des normalen Siebes für ein Faktorbasiselement q, für das man das Gitter berechnet, nur noch $|AB|/q$ Paare betrachten. Da man aber für kleine Faktorbasiselemente auf den Aufbau des Gitters verzichtet, muß zum Ausgleich dafür die Siebgrenzen etwas erhöht werden. Für die Praxis genügt dazu die Parameter $C_{min} = -C_{max}$ und D_{max} so zu wählen, daß

$$\frac{A_{max} \cdot B_{max}}{q_{min}} = C_{max} \cdot D_{max}$$

ist, wobei mit q_{min} das kleinste Faktorbasiselement bezeichnet wird, für das das Gitter aufgebaut wird.
Da die beiden Basisvektoren des aufgebauten Gitters, verschieden lang sind, ist es sinnvoll die Siebgrenze, die für den kürzeren Vektor gültig ist, um einen kleinen Faktor, z.B. zwei, größer zu wählen als die Grenze des anderen Basisvektors.

Der Korrekturfaktor

Der Korrekturfaktor KORRFAK wird benötigt, um die Ungenauigkeiten die durch das Sieben mit Logarithmen, den Verzicht auf kleine Faktorbasiselemente sowie auf Potenzen während der Siebphase entstehen, aufzufangen. Des Weiteren ist dieser Parameter dafür zuständig, daß Relationen mit zwei Large Primes in einer Zerlegung die Siebphase überstehen. Im folgenden werde ich den Korrekturfaktor nur für die rationale Seite betrachten, da der algebraische Faktor sich analog bestimmen läßt.
Da bis zu zwei Large Primes erlaubt sind, muß KORRFAK zumindest

$$2 \cdot \log(LPBOUND)$$

betragen. Wesentlich schwieriger ist die Frage zu lösen, um wieviel KORRFAK erhöht werden muß, um die Ungenauigkeiten abzufangen. Für Zerlegungen, in denen keine oder nur eine Large Prime vorkommt, muß KORRFAK nicht weiter erhöht werden, da $2 \cdot \log(LPBOUND)$ im Falle keiner Large Prime bzw. $\log(LPBOUND)$ im Falle einer Large Prime schon zuviel aufaddiert wurden. Für den Fall zweier Large Primes in einer Zerlegung gilt, daß wenn man KORRFAK zu hoch wählt, für zuviele Paare unnötigerweise Probedivision und gegebenenfalls Primzahltest sowie POLLARD_RHO durchgeführt werden müssen. Wählt man dagegen die Schranke zu niedrig, so verliert man einige gute Paare. Da aber im Korrekturfaktor schon zwei Large Primes enthalten sind, gehen vor allem solche Paare verloren, die in einer Zerlegung zwei Large Primes vom oberen Ende des Large Primes Intervall haben. Da aber große Large Primes in Relationen wesentlich seltener vorkommen als kleine, und somit seltener kombiniert werden können, hat der Verzicht auf diese Art von Relationen keine entscheidenden Auswirkung auf die Faktorisierung.
Für die Praxis empfehle ich eine kleine Erhöhung des Korrekturfaktors um etwa $\log(100)$, um den Verzicht auf kleine Faktorbasiselemente sowie auf Potenzen abzufangen. Insgesamt ergibt sich damit KORRFAK zu

$$\mathrm{KORRFAK} = \log\left(100 \cdot LPBOUND^2\right).$$

Tabelle 6.1 enthält eine mögliche Wahl der Parameter in Abhängigkeit von der Größe der zu faktorisierenden Zahl. Mit diesen Werten sollte eine Zerlegung der entsprechenden Zahlen möglich sein, wenn man mit Polynomen arbeitet, die nach den Algorithmen aus Kapitel 4 konstruiert worden sind.

Tabelle 6.1: geeignete Parameter

Dezimalstellen von n	$\lvert FB_R\rvert$	$\lvert FB_A\rvert$	LPBOUND	LPIBOUND	A_{max}
DLP–Variante					
30	750	2000	50000	20000	5000
40	1500	7500	$5 \cdot 10^5$	10^6	25000
50	2500	15000	$2 \cdot 10^6$	$5 \cdot 10^6$	$0.5 \cdot 10^6$
55	4000	21000	$4 \cdot 10^6$	$8 \cdot 10^6$	$1.2 \cdot 10^6$
QLP–Variante					
50	1800	12800	$3 \cdot 10^6$	$5 \cdot 10^6$	10^5
55	3000	14500	$5 \cdot 10^6$	$7.5 \cdot 10^6$	$2.5 \cdot 10^5$
60	4500	16500	$7.5 \cdot 10^6$	10^7	$6 \cdot 10^5$
65	6000	20000	10^7	$2.5 \cdot 10^7$	10^6
70	8000	24000	$2 \cdot 10^7$	$5 \cdot 10^7$	$1.5 \cdot 10^6$
75	10000	30000	$5 \cdot 10^7$	$7.5 \cdot 10^7$	$2.5 \cdot 10^6$
80	15000	35000	$6 \cdot 10^7$	10^8	$4 \cdot 10^6$
85	21000	45000	$8 \cdot 10^7$	10^8	$7 \cdot 10^6$
90	27000	65000	10^8	10^8	10^7
95	34000	90000	$2 \cdot 10^8$	10^8	$2 \cdot 10^7$
100	42000	115000	$5 \cdot 10^8$	10^8	$4 \cdot 10^7$
105	50000	150000	$7 \cdot 10^8$	$9 \cdot 10^8$	$6 \cdot 10^7$
110	65000	210000	10^9	10^9	$9 \cdot 10^7$
115	90000	290000	10^9	10^9	$1.1 \cdot 10^8$
120	125000	400000	$2 \cdot 10^9$	$2 \cdot 10^9$	$2 \cdot 10^8$
125	175000	550000	$2 \cdot 10^9$	$2 \cdot 10^9$	$3.5 \cdot 10^8$
130	250000	750000	$2 \cdot 10^9$	$2 \cdot 10^9$	$5 \cdot 10^8$

Kapitel 7

Beispiele von Faktorisierungen

In diesem Kapitel werde ich die Daten verschiedener von mir durchgeführter Faktorisierungen angeben. Es handelt sich dabei immer um die erste von mir durchgeführte Faktorisierung einer Zahl der entsprechenden Größenordnung.
Die Zahlen mit weniger als 100 Dezimalstellen wurden dabei dadurch erzeugt, daß man sich zwei Primzahlen vorgegebener Dezimalstellenanzahl zufällig generierte und miteinander multiplizierte.
Die Faktorisierung von Z_{107} wurde im Zusammenhang mit dem Lösen einer Diophantischen Gleichung benötigt. Z_{133} und Z_{152} stammen aus dem Cunningham–Projekt (siehe [BrRi92] bzw. [BLSTW83]). Bei RSA130 handelt es sich um die zum heutigen Zeitpunkt kleinste, nicht faktorisierte Zahl aus der RSA–Challenge Liste.
Die Parameter, die zur Faktorisierung dieser Zahlen verwendet wurden, muß nicht immer der von mir in Kapitel 6 vorgeschlagenen Parameterwahl entsprechen, da erst die aus diesen Faktorisierungen gewonnenen Erkenntnisse zu den vorgeschlagenen Parametern führten. Zur Lösung des lineraen Gleichungssystems wurde bis auf die 107–stellige Zahl stets die Structured Gauss Elimination verwendet.
Im letzten Abschnitt dieses Kapitels werde ich dann für verschiedene Zahlen einen Vergleich ziehen, welcher Schritt des NFS –Verfahrens welchen Anteil an der Gesamtlaufzeit des Algorithmus hat.

7.1 Faktorisierung einer 10–stelligen Zahl

1. $Z_{10} = 9140688449$
2. $m = 310$
3. $f(x) = 307x^3 - 54x^2 + 132x - 71$
4. die Faktorbasen

	FB_R	FB_A	FB_Q
Elementanzahl	50	112	10
größte Norm	229	521	605611571
Large Prime Grenze	2500	7500	-

5. Siebparameter

$$-500 \leq a \leq 500$$
$$1 \leq b \leq 30$$

6. Verteilung der 716 Relationen

alg \ rat	0 LP	1 LP
0 LP	141	125
1 LP	257	193

7. Verteilung der 350 Relationen nach dem Reduktionsschritt

alg \ rat	0 LP	1 LP
0 LP	141	64
1 LP	101	44

8. Verteilung der 242 Zykel

Full	Double
141	101

9. das lineare Gleichungssystem

   - 178×242 Matrix
   - durchschnittlich 1.59 Relationen pro Spalte
   - maximal 6 Relationen pro Spalte
   - 3421 Einträge modulo 2
   - durchschnittlich 14.14 Einträge pro Spalte
   - maximal 30 Einträge pro Spalte

10. die Berechnung der algebraischen Quadratwurzel

Stellen der Wurzelkoeffizienten ohne Reduktion mit Reduktion	 202 77
$\lvert S^+ \rvert$	44
$\lvert S^- \rvert$	40
Anzahl der *inert primes*	3
Größenordnung der *inert primes*	10^4
Potenz der *inert primes*	8

11. die Teiler

$$P_{05} \quad 95507$$
$$P_{05} \quad 95707$$

12. die Laufzeiten (in Hundertstel Mipssekunden)

Berechnung der Faktorbasis	960
Siebphase	5430
Zykelberechnung	3000
Lösen des LGS	42000
Wurzelberechnung	5838
insgesamt	56676

Dies entspricht 27 Sekunden auf einer 21–Mips Maschine.

7.2 Faktorisierung einer 20–stelligen Zahl

1. $Z_{20} = 12890677986877113421$

2. $m = 479353$

3. $f(x) = 117x^3 + 15957x^2 + 108173x + 120230$

4. die Faktorbasen

	FB_R	FB_A	FB_Q
Elementanzahl	200	534	10
größte Norm	1223	3571	605934949
Large Prime Grenze	20000	50000	-

5. Siebparameter

$$-500 \leq a \leq 500$$
$$1 \leq b \leq 60$$

6. Verteilung der 2393 Relationen

alg \ rat	0 LP	1 LP
0 LP	531	715
1 LP	483	664

7. Verteilung der 954 Relationen nach dem Reduktionsschritt

alg \ rat	0 LP	1 LP
0 LP	531	287
1 LP	79	57

8. Verteilung der 736 Zykel

Full	Double
531	205

9. das lineare Gleichungssystem

   - 748×736 Matrix
   - durchschnittlich 1.36 Relationen pro Spalte
   - maximal 4 Relationen pro Spalte
   - 11876 Einträge modulo 2
   - durchschnittlich 16.136 Einträge pro Spalte
   - maximal 30 Einträge pro Spalte

10. die Berechnung der algebraischen Quadratwurzel

Stellen der Wurzelkoeffizienten ohne Reduktion mit Reduktion	 202 77
$\|S^+\|$	181
$\|S^-\|$	179
Anzahl der *inert primes*	3
Größenordnung der *inert primes*	10^4
Potenz der *inert primes*	16

11. die Teiler

$$P_{09} \quad 169855297$$
$$P_{11} \quad 75892116493$$

12. die Laufzeiten (in Hundertstel Mipssekunden)

Berechnung der Faktorbasis	3960
Siebphase	18180
Zykelberechnung	11100
Lösen des LGS	67620
Wurzelberechnung (3 Versuche)	21210
insgesamt	122070

Dies entspricht 58 Sekunden auf einer 21–Mips Maschine.

7.3 Faktorisierung einer 30–stelligen Zahl

1. $Z_{30} = 884023530654951130225859246447$

2. $m = 91989277$

3. $f(x) = 1135670x^3 + 13245561x^2 + 20454546x + 31527126$

4. die Faktorbasen

	FB_R	FB_A	FB_Q
Elementanzahl	750	2065	10
größte Norm	5693	18313	613529569
Large Prime Grenze	60000	200000	-

5. Siebparameter

$$-7500 \leq a \leq 7500$$
$$1 \leq b \leq 500$$

6. Verteilung der 10970 Relationen

alg \ rat	0 LP	1 LP
0 LP	1525	2126
1 LP	3101	4218

7. Verteilung der 4669 Relationen nach dem Reduktionsschritt

alg \ rat	0 LP	1 LP
0 LP	1525	1185
1 LP	1136	823

8. Verteilung der 2979 Zykel

Full	Double
1525	1454

9. das lineare Gleichungssystem

   - 2979 $\times$ 2829 Matrix
   - durchschnittlich 1.77 Relationen pro Spalte
   - maximal 8 Relationen pro Spalte
   - 62793 Einträge modulo 2
   - durchschnittlich 21.08 Einträge pro Spalte
   - maximal 53 Einträge pro Spalte

10. die Berechnung der algebraischen Quadratwurzel

Stellen der Wurzelkoeffizienten	
ohne Reduktion	9119
mit Reduktion	1178
$\|S^+\|$	1005
$\|S^-\|$	953
Anzahl der *inert primes*	10
Größenordnung der *inert primes*	10^4
Potenz der *inert primes*	32

11. die Teiler

$$P_{15} \quad 499704385581371$$
$$P_{16} \quad 1769093000107357$$

12. die Laufzeiten (in Hundertstel Mipssekunden)

Berechnung der Faktorbasis	16680
Siebphase	468240
Zykelberechnung	66600
Lösen des LGS	116970
Wurzelberechnung	53718
insgesamt	722208

Dies entspricht 344 Sekunden auf einer 21–Mips Maschine.

7.4 Faktorisierung einer 40–stelligen Zahl

1. $Z_{40} = 5650376314972389829296691998260700479740 9$

2. $m = 46253073627$

3. $f(x) = 571025600x^3 + 10340614478x^2 + 15949656826x + 3360667045$

4. die Faktorbasen

	FB_R	FB_A	FB_Q
Elementanzahl	1500	7620	10
größte Norm	12553	76207	618276257
Large Prime Grenze	150000	1000000	-

5. Siebparameter

$$-25000 \leq a \leq 25000$$
$$1 \leq b \leq 2500$$

6. Verteilung der 35631 Relationen

rat alg	0 LP	1 LP
0 LP	5232	9847
1 LP	7407	13145

7. Verteilung der 17432 Relationen nach dem Reduktionsschritt

rat alg	0 LP	1 LP
0 LP	5232	7074
1 LP	2265	2861

8. Verteilung der 11572 Zykel

Full	Double
5232	6340

9. das lineare Gleichungssystem

- 11572×9134 Matrix
- durchschnittlich 1.80 Relationen pro Spalte
- maximal 9 Relationen pro Spalte

- 285767 Einträge modulo 2
- durchschnittlich 24.69 Einträge pro Spalte
- maximal 73 Einträge pro Spalte

10. die Berechnung der algebraischen Quadratwurzel

Stellen der Wurzelkoeffizienten ohne Reduktion mit Reduktion	 26773 3662
$\|S^+\|$	2201
$\|S^-\|$	2105
Anzahl der *inert primes*	10
Größenordnung der *inert primes*	10^6
Potenz der *inert primes*	64

11. die Teiler

P_{20} 76969789585536373499
P_{21} 734103126096393691091

12. die Laufzeiten (in Hundertstel Mipssekunden)

Berechnung der Faktorbasis	68040
Siebphase	3456030
Zykelberechnung	382200
Lösen des LGS	303450
Wurzelberechnung	231021
insgesamt	4440741

Dies entspricht etwa 35 Minuten auf einer 21–Mips Maschine.

7.5 Faktorisierung einer 50–stelligen Zahl

1. $Z_{50} = 25687932972608082604877595262293865434242386373017$
2. $m = 13507766361970$
3. $f(x) = 10422659230x^3 - 1512842436406x^2 - 5686469940671x - 1485447289713$
4. die Faktorbasen

	FB_R	FB_A	FB_Q
Elementanzahl	2500	14707	10
größte Norm	22307	163841	605934949
Large Prime Grenze	300000	1500000	-

5. Siebparameter

$$-500000 \leq a \leq 500000$$
$$1 \leq b \leq 5000$$

6. Verteilung der 56029 Relationen

alg \ rat	0 LP	1 LP
0 LP	8082	17774
1 LP	9859	20314

7. Verteilung der 27271 Relationen nach dem Reduktionsschritt

alg \ rat	0 LP	1 LP
0 LP	8082	12339
1 LP	2777	4073

8. Verteilung der 18126 Zykel

Full	Double
8082	10044

9. das lineare Gleichungssystem

   - 18026×17221 Matrix
   - durchschnittlich 1.77 Relationen pro Spalte
   - maximal 12 Relationen pro Spalte
   - 500583 Einträge modulo 2
   - durchschnittlich 27.77 Einträge pro Spalte
   - maximal 113 Einträge pro Spalte

10. die Berechnung der algebraischen Quadratwurzel

Stellen der Wurzelkoeffizienten	
ohne Reduktion	83149
mit Reduktion	10371
$\|S^+\|$	5651
$\|S^-\|$	5443
Anzahl der *inert primes*	7
Größenordnung der *inert primes*	10^6
Potenz der *inert primes*	256

11. die Teiler

$$P_{24} \quad 129835477251149736243011$$
$$P_{27} \quad 197849875214908623387586547$$

12. die Laufzeiten (in Mipssekunden)

Berechnung der Faktorbasis	1452
Siebphase	364140
Zykelberechnung	8703
Lösen des LGS	10080
Wurzelberechnung (3 Versuche)	54285
insgesamt	438660

Dies entspricht etwa 348 Minuten auf einer 21–Mips Maschine.

7.6 Faktorisierung einer 55–stelligen Zahl

1. $Z_{55} = 3186689262657991595114095972260935292854216673468006443$
2. $m = 115843343213940$
3.

$$f(x) = 2049870705372\ x^3 - 4354314195329\ x^2 + 7827380602539\ x + 24987736229183$$

4. die Faktorbasen

	FB_R	FB_A	FB_Q
Elementanzahl	3000	15091	10
größte Norm	27449	163841	617650259
Large Prime Grenze	500000	1500000	-

5. Siebparameter

$$-400000 \leq a \leq 400000$$
$$1 \leq b \leq 18000$$

6. Verteilung der 177103 Relationen

alg / rat	0 LP	1 LP	2 LP
0 LP	4242	15361	13365
1 LP	12417	44530	38295
2 LP	6705	23299	18889

7. Verteilung der 140170 Relationen nach dem Reduktionsschritt

alg / rat	0 LP	1 LP	2 LP
0 LP	4242	14890	12594
1 LP	10176	35646	29674
2 LP	4702	15760	12486

8. Verteilung der 64064 Zykel

Full	Double	Quadruple
4242	19508	40314

9. das lineare Gleichungssystem Für das Gleichungssystem wurden 40000 Relationen verwendet, was in etwa doppelt so viele, wie benötigt waren.

   - 40896×18105 Matrix
   - 2374112 Einträge modulo 2
   - durchschnittlich 58.05 Einträge pro Spalte
   - maximal 214 Einträge pro Spalte

10. die Berechnung der algebraischen Quadratwurzel

Stellen der Wurzelkoeffizienten	
ohne Reduktion	204367
mit Reduktion	18732
Anzahl der *inert primes*	15
Größenordnung der *inert primes*	10^5
Potenz der *inert primes*	256

11. die Teiler

$$P_{27} \quad 462962907440001211551863869$$
$$P_{28} \quad 6883249632846618999754253447$$

12. die Laufzeiten (in Mipsminuten)

Berechnung der Faktorbasis	22.25
Siebphase	29073.13
Zykelberechnung	1293.70
Lösen des LGS	955.89
Wurzelberechnung	1057.5
insgesamt	32402.5

Dies entspricht etwa 25.72 Stunden auf einer 21–Mips Maschine.

7.7 Faktorisierung einer 60–stelligen Zahl

1. $Z_{60} = 798851513104416932353630634050150408748076412209030336349031$
2. $m = 9454019974160380$
3.

$$\begin{aligned} f(x) = \quad & 945401997416 \; x^3 + 345851019291160 \; x^2 \\ & + 3518759467845011 \; x + 3842009737228851 \end{aligned}$$

4. die Faktorbasen

	FB_R	FB_A	FB_Q
Elementanzahl	5000	19924	10
größte Norm	48611	224737	613529569
Large Prime Grenze	500000	2500000	-

5. Siebparameter

$$-2000000 \leq a \leq 2000000$$
$$1 \leq b \leq 15000$$

6. Verteilung der 142843 Relationen

alg \ rat	0 LP	1 LP
0 LP	6684	12471
1 LP	23674	44536
2 LP	19918	35560

7. Verteilung der 63508 Relationen nach dem Reduktionsschritt

alg \ rat	0 LP	1 LP
0 LP	6684	9516
1 LP	13004	18921
2 LP	6484	8899

8. Verteilung der 25173 Zykel

Full	Double	Triple
6684	12378	6111

9. das lineare Gleichungssystem

   - 25173×24939 Matrix
   - durchschnittlich 4.05 Relationen pro Spalte
   - maximal 40 Relationen pro Spalte
   - 1345986 Einträge modulo 2
   - durchschnittlich 53.47 Einträge pro Spalte
   - maximal 367 Einträge pro Spalte

10. die Berechnung der algebraischen Quadratwurzel

Stellen der Wurzelkoeffizienten	
ohne Reduktion	407934
mit Reduktion	36849
$\|S^+\|$	22994
$\|S^-\|$	22256
Anzahl der *inert primes*	50
Größenordnung der *inert primes*	10^6
Potenz der *inert primes*	128

11. die Teiler

P_{29} 49354632530921060679237683627

P_{32} 16185947947316399402298813958453

12. die Laufzeiten (in Mipsminuten)

Berechnung der Faktorbasis	38.15
Siebphase	108197.57
Zykelberechnung	475.1
Lösen des LGS	943.2
Wurzelberechnung (2 Versuche)	4472.73
insgesamt	114127

Dies entspricht etwa 90.5 Stunden auf einer 21–Mips Maschine.

7.8 Faktorisierung einer 65–stelligen Zahl

1. $Z_{65} =$ 13568058583965491335510161033083294839111037053087670936678126323

2. $m = 111112911289297391$

3.

$$\begin{aligned} f(x) = \quad & 375844090766034 \; x^3 \; +2868020802012825 \; x^2 \\ & -1052876551716967 \; x \; +2092686118884532 \end{aligned}$$

mit $f(m) = 38 \cdot n$

4. die Faktorbasen

	FB_R	FB_A	FB_Q
Elementanzahl	6000	18752	10
größte Norm	59359	212369	613529569
Large Prime Grenze	750000	2500000	-

5. Siebparameter

$$-3000000 \leq a \leq 3000000$$
$$1 \leq b \leq 40000$$

6. Verteilung der 181328 Relationen

alg \ rat	0 LP	1 LP	2 LP
0 LP	2973	9510	8167
1 LP	12850	40861	34527
2 LP	10417	33974	28049

7. Verteilung der 102918 Relationen nach dem Reduktionsschritt

alg \ rat	0 LP	1 LP	2 LP
0 LP	2973	8575	6717
1 LP	8806	25014	19032
2 LP	5163	15048	11590

8. Verteilung der 27664 Zykel

Full	Double	quadruple
2973	5326	19365

9. das lineare Gleichungssystem

   - 27485×24756 Matrix [1]
   - 4674597 Einträge modulo 2
   - durchschnittlich 18.01 Relationen pro Spalte
   - maximal 214 Relationen pro Spalte
   - durchschnittlich 170.08 Einträge pro Spalte
   - maximal 778 Einträge pro Spalte

10. die Berechnung der algebraischen Quadratwurzel

Stellen der Wurzelkoeffizienten ohne Reduktion mit Reduktion	 1640173 71079
$\|S^+\|$	81418
$\|S^-\|$	81254
Anzahl der *inert primes*	14
Größenordnung der *inert primes*	10^5
Potenz der *inert primes*	1024

[1]alle Zykel mit mehr als 100 Paaren (a, b) wurden ignoriert

11. die Teiler

$$\begin{array}{ll} P_{29} & 27578212350327442415819829667 \\ P_{36} & 491984701967254040530716432196537969 \end{array}$$

12. die Laufzeiten (in Mipsstunden)

Berechnung der Faktorbasis	0.47
Siebphase	4994.42
Zykelberechnung	31.54
Lösen des LGS	53.15
Wurzelberechnung	235.76
insgesamt	5315.34

Dies entspricht etwa 253 Stunden auf einer 21–Mips Maschine.

7.9 Faktorisierung einer 71–stelligen Zahl

1. $Z_{71} = 17529234485759798885614651375503993108734254574339382 33\backslash$
   1210841812819361

2. $m = 865422599729859616$
3.

$$\begin{array}{rlrl} f(x) = & 27044456241558080 & x^3 & +140873032291901555 \quad x^2 \\ & +174915571868806256 & x & -245234314315520 95 \end{array}$$

4. die Faktorbasen

	FB_R	FB_A	FB_Q
Elementanzahl	7500	24920	10
größte Norm	76207	287117	644252327
Large Prime Grenze	20000000	50000000	-

5. Siebparameter

$$-2000000 \leq a \leq 2000000$$
$$1 \leq b \leq 115000$$

6. Verteilung der 3252596 Relationen

alg \ rat	0 LP	1 LP	2 LP
0 LP	2766	25625	60060
1 LP	30340	286642	664888
2 LP	69327	646538	1466410

7. Verteilung der 1820315 Relationen nach dem Reduktionsschritt

alg \ rat	0 LP	1 LP	2 LP
0 LP	2766	22552	48104
1 LP	22703	187574	397039
2 LP	43363	356470	739744

8. Verteilung der 587642 Zykel

Full	Double	Quadruple
2766	5565	579311

9. das lineare Gleichungssystem

   - 42936 $\times$ 32443 Matrix
   - 6898636 Einträge modulo 2
   - durchschnittlich 7.563 Relationen pro Spalte
   - maximal 25 Relationen pro Spalte
   - durchschnittlich 160.67 Einträge pro Spalte
   - maximal 306 Einträge pro Spalte

10. die Berechnung der algebraischen Quadratwurzel

Stellen der Wurzelkoeffizienten ohne Reduktion mit Reduktion	 1918990 83834
$\|S^+\|$	87562
$\|S^-\|$	87524
Anzahl der *inert primes*	28
Größenordnung der *inert primes*	10^6
Potenz der *inert primes*	512

11. die Teiler

P_{35}	39478938423253105368951289389883223
P_{36}	444014838946000509735726958027761607

12. die Laufzeiten (in Mipsstunden)

Berechnung der Faktorbasis	0.65
Siebphase	26234.9
Zykelberechnung	58.06
Lösen des LGS	108.06
Wurzelberechnung (3 Versuche)	1569.41
insgesamt	27971.0

Dies entspricht etwa 55.5 Tagen auf einer 21–Mips Maschine.

7.10 Faktorisierung einer 75–stelligen Zahl

1. Z_{75} = 285047312769308978048559762663660354662988887201554033\
463110001262549589043

2. $m = 12989727254543599862$
3.

$$f(x) = \begin{array}{rlrl} 130051945773428835 & x^3 & +509308602799278565 & x^2 \\ -2065521180992881273 & x & -2155403748668962371 & \end{array}$$

4. die Faktorbasen

	FB_R	FB_A	FB_Q
Elementanzahl	10000	24941	10
größte Norm	104729	287117	614802269
Large Prime Grenze	50000000	75000000	-

5. Siebparameter

$$-6000000 \leq a \leq 6000000$$
$$1 \leq b \leq 80000$$

6. Verteilung der 3652233 Relationen

alg \ rat	0 LP	1 LP	2 LP
0 LP	1710	17924	47143
1 LP	22484	244382	637425
2 LP	68620	736567	1875978

7. Verteilung der 1022120 Relationen nach dem Reduktionsschritt

alg \ rat	0 LP	1 LP	2 LP
0 LP	1710	12009	23606
1 LP	18305	120724	190787
2 LP	29482	214006	411491

8. Verteilung der 147432 Zykel

Full	Double	Quadruple
1710	949	144773

9. das lineare Gleichungssystem

   - 38983×34954 Matrix[2]
   - durchschnittlich 49.21 Relationen pro Spalte
   - maximal 105 Relationen pro Spalte
   - 18578126 Einträge modulo 2
   - durchschnittlich 476 Einträge pro Spalte
   - maximal 886 Einträge pro Spalte

10. die Berechnung der algebraischen Quadratwurzel

Stellen der Wurzelkoeffizienten ohne Reduktion mit Reduktion	 8889076 237347
$\|S^+\|$	386016
$\|S^-\|$	385388
Anzahl der *inert primes*	39
Größenordnung der *inert primes*	10^6
Potenz der *inert primes*	1024

[2]alle Zykel mit mehr als 105 Paaren (a, b) wurden ignoriert

11. die Teiler

$$P_{36} \quad 314749722256519077689398475430484051$$
$$P_{39} \quad 905631657831923932422706731329431007393$$

12. die Laufzeiten (in Mipsstunden)

Berechnung der Faktorbasis	0.28
Siebphase	36187.8
Zykelberechnung	170.51
Lösen des LGS	235.42
Wurzelberechnung (2 Versuche)	13708.11
insgesamt	50302.1

Dies entspricht etwa 100 Tagen auf einer 21–Mips Maschine.

7.11 Faktorisierung einer 80–stelligen Zahl

1. $Z_{80} = 45820458660007135919396875046356046256139474684514187 9\backslash$
   $79017934209961585590174637$

2. $m = 106165866563988$
3.

$$\begin{aligned} f(x) = \quad & 3397307730 \; x^5 && -1057996443398 \; x^4 \\ & -15163372826372 \; x^3 && -24058949101584 \; x^2 \\ & +2083823133910 \; x && +16602672539125 \end{aligned}$$

4. die Faktorbasen

	FB_R	FB_A	FB_Q
Elementanzahl	15000	35029	10
größte Norm	163841	414977	610681597
Large Prime Grenze	75000000	100000000	-

5. Siebparameter (Lattice Sieve)

$$-4000 \leq c \leq 4000$$
$$1 \leq d \leq 9000$$

für q vom 5001. bis 35029. algebraischen Faktorbasiselement

6. Verteilung der 5609241 Relationen

alg \ rat	0 LP	1 LP	2 LP
0 LP	5352	33861	38552
1 LP	73319	477537	566878
2 LP	276294	1839468	2297980

7. Verteilung der 1991341 Relationen nach dem Reduktionsschritt

alg \ rat	0 LP	1 LP	2 LP
0 LP	5352	23496	21786
1 LP	49240	223884	213699
2 LP	136825	631804	685255

8. Verteilung der 446321 Zykel

Full	Double	Quadruple
5352	7303	433666

9. das lineare Gleichungssystem

   - 107007 $\times$ 50042 Matrix [3]
   - durchschnittlich 37.46 Relationen pro Spalte
   - maximal 100 Relationen pro Spalte
   - 38420279 Einträge modulo 2
   - durchschnittlich 359 Einträge pro Spalte
   - maximal 3016 Einträge pro Spalte

10. die Berechnung der algebraischen Quadratwurzel

Stellen der Wurzelkoeffizienten ohne Reduktion mit Reduktion	 2940318 183022
$\|S^+\|$	166965
$\|S^-\|$	168945
Anzahl der *inert primes*	35
Größenordnung der *inert primes*	10^5
Potenz der *inert primes*	1024

[3]alle Zykel mit mehr als 100 Paaren (a, b) wurden ignoriert

11. die Teiler

$$\begin{array}{ll} P_{32} & 94149729925136579880658431514781 \\ P_{48} & 486676474764626529591760336621189059816202666577 \end{array}$$

12. die Laufzeiten (in Mipsstunden)

Berechnung der Faktorbasis	1.87
Siebphase	82380.25
Zykelberechnung	240.67
Lösen des LGS	298.41
Wurzelberechnung (3 Versuche)	6201.24
insgesamt	89122.4

Dies entspricht etwa 176 Tagen auf einer 21–Mips Maschine.

7.12 Faktorisierung einer 107–stelligen Zahl

Diese Zahl wurde in Gemeinschaftsarbeit mit Marije Huizing, Arjen Lenstra, Paul Leyland und Peter Montgomery faktorisiert. Die Siebphase wurde von allen fünf beteiligten Personen gemeinsam durchgeführt. Arjen Lenstra hat die Relationen gesammelt, die Zykel berechnet und die Matrix aufgebaut. Marije Huizing und Peter Montgomery lösten das lineare Gleichungssystem am CWI in Amsterdam und bestimmten die Teiler von Z_{107}.

1. $Z_{107} = 190169992682135463950623295712596155380467950821248119\backslash$
   $832894120815219769600778796964721272826534108658815 31$

2. $m = 1553763704277837613$
3.

$$\begin{array}{rrlrl} f(x) = & -2216129448497968 & x^5 & -984939804547401 & x^4 \\ & -3802452283948140 & x^3 & +805460504833795 & x^2 \\ & -1875319610106137 & x & +2099989527510798 & \end{array}$$

4. die Faktorbasen

	FB_R	FB_A
Elementanzahl	50000	150000
größte Norm	611953	2012711
Large Prime Grenze	10^9	10^9

5. Siebparameter (Lattice Sieve)

$$-4200 \le c \le 4200$$
$$1 \le d \le 9600$$

   für q vom 60001. bis 150000. algebraischen Faktorbasiselement

6. Verteilung der 34150634 Relationen.

   Hier ist auch zu berücksichtigen, daß Peter Montgomery zeitweise mit größeren Faktorbasen gearbeitet hat, was dazu führte, daß Relationen mit bis zu sechs algebraischen und bis zu vier rationalen Large Primes entdeckt wurden. Diese Relationen werden in zwei Typen unterteilt. Zum Typ A gehören die Relationen mit genau drei Large Primes in einer der beiden Zerlegungen und höchstens drei in der anderen. Zum Typ B gehören alle Relationen mit mehr als drei Large Primes auf einer der beiden Seiten.

alg \ rat	0 LP	1 LP	2 LP
0 LP	14711	134538	244743
1 LP	225544	2056083	3751722
2 LP	822409	7523418	13821690

   Relationen vom Typ A: 3553854

   Relationen vom Typ B: 2001922

7. Verteilung der 5530972 Relationen nach dem Reduktionsschritt

alg \ rat	0 LP	1 LP	2 LP
0 LP	14711	72768	90602
1 LP	116836	576068	722332
2 LP	268966	1334584	1685026

   Relationen vom Typ A: 430112

   Relationen vom Typ B: 218967

8. Diese Relationen ergaben 775107 Zykel

9. die Teiler

   P_{53} 11613490967547337524883597328386942639346032555901449

   P_{55} 1637492061719815650585855997104944256456021560117946019

10. Die Gesamtlaufzeit wurde nicht exakt bestimmt. Sie beträgt etwa 147 Mipsjahre, was 7 Jahren auf einer 21–Mips Maschine entspricht.

7.13 Weltrekord RSA130 (unvollendet)

Die Zahl RSA130 ist eine von der Firma RSA herausgegebene Zahl, die aus zwei, etwa gleich großen Primteilern aufgebaut ist. Die Faktorisierung dieser Zahl wird in Gemeinschaftsarbeit mit Marije Huizing, Arjen Lenstra, Paul Leyland und Peter Montgomery faktorisiert. Die Relationensuche dauert derzeit noch an. Es werden etwa 45 Millionen Relationen zur Erzeugung ausreichend vieler Zykel benötigt.

1. RSA130 = 18070820886874048059516561644059055662781025167694013491\
   701270214500566625402440483873411275908123033717818879\
   66563182013214880557

2. $m = 125744111684180059804 68$

3. $$\begin{aligned} f(x) = \quad & 5748302248738405200\ x^5 & +9882261917482286102\ x^4 \\ & -13392499389128176685\ x^3 & +16875252458877684989\ x^2 \\ & +3759900174855208738\ x & -46769930553931905995 \end{aligned}$$

4. die Faktorbasen

	FB_R	FB_A
Elementanzahl	250000	1000000
größte Norm	3497861	15485863
Large Prime Grenze	$2 \cdot 10^9$	$2 \cdot 10^9$

5. Siebparameter (Lattice Sieve)

$$-10000 \leq c \leq 10000$$
$$1 \leq d \leq 12000$$

   für q vom 50001. bis 1000000. algebraischen Faktorbasiselement

6. Die Gesamtlaufzeit wird auf etwa 1100 Mipsjahre geschätzt, was 52 Jahren auf einem 21 Mips–Rechner entspricht.

7.14 Faktorisierung einer 133–stelligen speziellen Zahl

Die Faktorisierung dieser Zahl wurde mit meiner ersten Implementierung des Lattice Sieve Algorithmus durchgeführt. Die Zerlegung von Z_{133} wurde ohne die Quadruple Large Prime Variante durchgeführt. Die Berechnung der algebraischen Quadratwurzeln wurde ohne den in Kapitel 5.2.1 beschriebene Reduktion durchgeführt. Stattdessen wurde die in [BuLoZa93] beschriebene schwächere Reduktion verwendet.

1. $Z_{133} = 4362232530202016608116950834542112097947919092693930724\backslash$
   $9279375370109414452149539140120565249995711637236858619\backslash$
   99536219765430952971290

2. $m = 3^{56} = 523347633027360537213511521$

3. $$f(x) = x^5 + 9$$

   mit $f(m) = 9 \cdot n$

4. die Faktorbasen

	FB_R	FB_A	FB_Q
Elementanzahl	75000	74952	25
größte Norm	951161	951109	-
Large Prime Grenze	750000	2500000	-

5. Siebparameter (Lattice Sieve)

$$-10000 \leq c \leq 10000$$
$$1 \leq d \leq 5250$$

6. Verteilung der 1634475 Relationen

alg \ rat	0 LP	1 LP
0 LP	73798	184864
1 LP	344560	1031253

7. Verteilung der 152901 Zykel

Full	Double
73798	79103

8. die Berechnung der algebraischen Quadratwurzel

Stellen der Wurzelkoeffizienten Anzahl der *inert primes*	135500 105
Größenordnung der *inert primes*	$1.1 \cdot 10^5$
Potenz der *inert primes*	256

9. die Teiler

P_1	2
P_1	3
P_1	3
P_1	5
P_3	557
P_7	1107553
P_{16}	8207398122145081
P_{41}	13857905391453292485606236633776204574597
P_{67}	6217073567621646188942987882827287720857305423132773876\ 341378217457

10. die Laufzeiten (in Mipstagen)

Berechnung der Faktorbasis	0.19
Siebphase	41010.72
Zykelberechnung	12.44
Lösen des LGS	21.09
Wurzelberechnung (4 Versuche)	80.75
insgesamt	41125.2

Dies entspricht etwa 5 Jahren und 133 Tagen auf einer 21–Mips Maschine.

7.15 Faktorisierung einer 152–stelligen speziellen Zahl

Diese Zahl kann derzeit nur deshalb faktorisiert werden, weil sie eine spezielle Form hat und man deshalb ein Polynom mit betragsmäßig sehr kleinen Koeffizienten finden kann.

1. Z_{152} = 3135421367873363591116465091247905728723238010813326814\
   3361865784085839771102774695632601950948771908816812831\
   5529438013380106116295299745536098816464 2

2. $m = 1192533292512492016559195008117$
3. $$f(x) = 13x^5 + 1$$
4. die Faktorbasen

	FB_R	FB_A	FB_Q
Elementanzahl	65000	75062	15
größte Norm	1628553	1902188	628416353
Large Prime Grenze	50000000	75000000	-

5. Siebparameter

$$-6000000 \leq a \leq 6000000$$
$$1 \leq b \leq 1500000$$

6. Verteilung der 5598465 Relationen

alg \ rat	0 LP	1 LP	2 LP
0 LP	12363	93447	200384
1 LP	76365	574882	1236337
2 LP	137141	1039214	2228332

7. Verteilung der 2883273 Relationen nach dem Reduktionsschritt

alg \ rat	0 LP	1 LP	2 LP
0 LP	12363	77261	143005
1 LP	57930	360798	668315
2 LP	82919	519433	961249

8. Verteilung der 622325 Zykel

Full	Double	Quadruple
12363	11951	598011

9. das lineare Gleichungssystem

- 152268×140081 Matrix [4]
- 71660236 Einträge modulo 2
- durchschnittlich 26.43 Relationen pro Spalte
- maximal 90 Relationen pro Spalte
- durchschnittlich 470.62 Einträge pro Spalte
- maximal 676 Einträge pro Spalte

[4] alle Zykel mit mehr als 100 Paaren (a, b) wurden ignoriert

10. die Berechnung der algebraischen Quadratwurzel

Stellen der Wurzelkoeffizienten ohne Reduktion mit Reduktion	 9607791 690173
$\|S^+\|$	1360338
$\|S^-\|$	1356590
Anzahl der *inert primes*	108
Größenordnung der *inert primes*	$2 \cdot 10^6$
Potenz der *inert primes*	1024

11. die Laufzeiten (in Mipstagen)

Berechnung der Faktorbasis	0.06
Siebphase	23073.1
Zykelberechnung	29.99
Lösen des LGS	602.14
Wurzelberechnung (2 Versuche)	3965.52
insgesamt	27670.9

Dies entspricht etwa 1318 Tagen auf einer 21–Mips Maschine.

7.16 Prozentuale Verteilung der Laufzeit

In diesem Abschnitt möchte ich auf das Verhältnis der Laufzeiten der drei laufzeitintensiven Schritte

1. Relationenberechnung (REL)

2. Aufstellen und Lösen des linearen Gleichungssystems modulo 2 (LGS)

3. Berechnung der algebraischen Quadratwurzel (SQRT)

eingehen.
Tabelle 7.1 zeigt in Abhängigkeit von der Größe der faktorisierten Zahl, wie sich die Laufzeiten prozentual auf die einzelnen Schritte verteilen.
Die Tabelle zeigt deutlich, daß die Laufzeit des *Number Field Sieve* ab 30–stelligen Zahlen von der Siebphase dominiert wird. Im Falle der 75–stelligen Zahl tritt die Berechnung der Quadratwurzel stärker in Erscheinung. Dies hängt damit zusammen, daß bei dieser Faktorisierung riesige Zykellängen verwendet wurden, und die Zykel nicht durch vermehrtes Sieben reduziert worden sind.

Tabelle 7.1: Verteilung der Laufzeit

n	Laufzeit in Prozent		
	REL	LGS	SQRT
Z_{10}	9.6	79.4	10.3
Z_{20}	14.9	64.5	17.4
Z_{30}	64.8	25.4	7.4
Z_{40}	77.8	15.4	5.2
Z_{50}	83.0	4.28	12.3
Z_{55}	89.7	6.9	3.3
Z_{60}	94.8	1.2	3.9
Z_{65}	93.9	1.6	4.4
Z_{71}	93.8	0.59	5.6
Z_{75}	71.9	0.81	27.2
Z_{80}	92.4	6.96	0.6

Anhang A

Fundamentale zahlentheoretische Algorithmen

In diesem Kapitel werde ich Algorithmen beschreiben, die nicht nur im *Number Field Sieve* bzw. in Faktorisierungsalgorithmen eine Rolle spielen sondern auch in vielen anderen Bereichen der Computeralgebra Anwendung finden.

A.1 Schnelle Exponentiation

Sei g ein Element aus einer beliebigen multiplikativen Gruppe G. Das Berechnen der d-ten Potenz, $d \in \mathbb{N}$, von g ist eine Aufgabe, die unter Anwendung des naiven Algorithmus, $d-1$ Multiplikationen erfordert. Die schnelle Exponentiation ist eine Potenzierungsmethode, die im Höchstfall

$$2 \cdot \lfloor \log(d) \rfloor$$

Multiplikationen und

$$\lfloor \log(d) \rfloor$$

Binärshifts benötigt. Zur Erklärung des Prinzips betrachten wir die Binärdarstellung des Exponenten d:

$$d = d_0 + d_1 \cdot 2 + \ldots + d_{\lfloor \log(d) \rfloor} \cdot 2^{\lfloor \log(d) \rfloor},\ d_i \in \{0, 1\}.$$

Somit gilt

$$\begin{aligned} g^d &= g^{d_0 + d_1 \cdot 2 + \ldots + d_{\lfloor \log(d) \rfloor} \cdot 2^{\lfloor \log(d) \rfloor}} \\ &= g^{d_0} \cdot g^{d_1 \cdot 2} \cdot \ldots \cdot g^{d_{\lfloor \log(d) \rfloor} \cdot 2^{\lfloor \log(d) \rfloor}}. \end{aligned} \tag{A.1}$$

Dies bedeutet, g^d läßt sich durch sukzessives Aufmultiplizieren geeigneter Zweierpotenzen von g berechnen. Geeignet sind dabei die Potenzen 2^i mit $d_i = 1$. Um g^d auf diese Weise berechnen zu können, benötigt man also $\lfloor \log(d) \rfloor$ Multiplikationen zum Multiplizieren der Zweierpotenzen von g und ebensoviele Multiplikationen, um die einzelnen Zweierpotenzen zu berechnen. Bei der letzten Abschätzung wurde benutzt, daß die $(i+1)$-te Zweierpotenz aus der i-ten durch Quadrierung, also durch eine Multiplikation hervorgeht. Die Binärshifts werden benötigt, um die Koeffizienten d_i der Binärdarstellung von d zu berechnen.

Algorithmus A.1 (schnelle Exponentiation)

FASTEXPO

EINGABE: $g \in G$, G multiplikative Gruppe, $d \in \mathbb{N}$
AUSGABE: g^d

(1)	$gd := 1$	/* neutrales Element */
(2)	**while** $(d > 0)$ **do**	/* arbeite d bitweise ab */
(3)	**if** (d ungerade) **then**	/* es muß mit g^{2^i} multipliziert werden */
(4)	$gd := gd \cdot g$	/* siehe (A.1) */
(5)	$g := g^2$	/* nächste Zweierpotenz */
(6)	**od**	
(7)	**return** (gd)	

In manchen Gruppen, z.B. Restklassengruppen, können in der Praxis zur Vermeidung großer Zwischenergebnisse die Ergebnisse der Multiplikation durch Restbildung mit dem Modul in eine gewünschte Darstellungsform gebracht werden[1]. Deshalb muß nach der Multiplikation in Schritt (5) bzw. nach der Quadrierung in Schritt (6) noch die Reduktion mit Hilfe des Moduls vorgenommen werden.

A.2 Näherungen für die Wurzeln ganzer Zahlen

In vielen Anwendungen, siehe z.B. Bemerkung 2.2 oder Gleichung (4.1), ist es notwendig, die der d-ten Wurzel, $d \in \mathbb{N}$, einer positiven ganzen Zahl z nächstgelegene ganze Zahl $[\sqrt[d]{z}]$ zu berechnen. Steht eine lange Gleitkommaarithmetik, wie z.B. die *libF* ([Pa93]), zur Verfügung, so läßt sich wegen der Identität

$$\sqrt[d]{z} = z^{1/d} = \exp\left(\log(z)/d\right),$$

die Wurzel einfach durch Verwendung der Exponentialfunktion und des natürlichen Logarithmus berechnen.
Kann man nicht auf eine solche Arithmetik zurückgreifen, so läßt sich das gewünschte Ergebnis auch allein mit Integeroperationen errechnen. Dabei geht man, wie bei der binären Suche vor. Zuerst werden eine untere (US) und eine obere Schranke (OS) festgelegt, wobei 1 bzw. z die trivialen Schranken sind. Dann berechnet man den ganzzahligen Wert MIT, der dem arithmetischen Mittel von US und OS am nächsten kommt. Ist MIT^d größer als z, so liegt die gesuchte Wurzel unterhalb von MIT. Die obere Schranke OS wird deshalb auf $MIT - 1$ heruntergesetzt. Ist MIT^d dagegen kleiner als z, so liegt die gesuchte Wurzel oberhalb von MIT. US wird dann auf $MIT + 1$ hochgesetzt. Die Suche wird im neuen Intervall [US, OS] fortgesetzt. Auf diese Weise wird Schritt um Schritt das Intervall, in dem der gesuchte Wert liegt, in etwa halbiert, bis nur $[\sqrt[d]{z}]$ übrig bleibt.

[1] z.B. $\mathbb{Z}/p\mathbb{Z}$, $\mathbb{Z}[x]/f(x)$

Algorithmus A.2 (Integerwurzel)

DROOTZ

EINGABE: $z \in \mathbb{N}$, $d \in \mathbb{N}$
AUSGABE: $\lfloor \sqrt[d]{z} \rfloor$

```
US := 1; OS := z                 /* Initialisierung */
while (US < OS - 1) do           /* bis nur noch US und OS in
                                    Frage kommen */
  MIT := ⌊(US + OS)/2⌋           /* Mittelwert */
  POT := MIT^d                   /* d–te Potenz des Mittelwertes */
  if (POT = z) then              /* Wurzel gefunden */
    return (MIT)
  if (POT > z) then
    OS := MIT - 1                /* Obere Schranke verschieben */
  else
    US := MIT                    /* Untere Schranke verschieben */
  fi
od
if (OS^d > z) then               /* OS zu groß ? */
  return (US)
else
  return (OS)
fi
```

A.3 Das Legendresymbol

Im *Number Field Sieve* benötigen wir das Legendresymbol zur Berechnung der quadratischen Integer, die in (2.34) auf Seite 33 in Kapitel 2.3.2 beschrieben sind. Das Legendresymbol $\left(\frac{a}{p}\right)$ gibt an, ob eine ganze Zahl a ein quadratischer Rest modulo einer Primzahl p ist oder nicht. Genau ist das Legendre Symbol wie folgt definiert;

Definition A.3 (Legendresymbol) ([Ch93, Kapitel 1.4.2])
Sei p eine ungerade Primzahl und $a \in \mathbb{Z}$, dann gilt

$$\left(\frac{a}{p}\right) := \begin{cases} 1, & \text{wenn } x^2 \equiv a \pmod p \text{ Lösung hat} \\ -1, & \text{wenn } x^2 \equiv a \pmod p \text{ keine Lösung hat} \\ 0, & \text{wenn } p|a \end{cases} \tag{A.2}$$

□

Die Verallgemeinerung des Legendresymbols ist das Jacobisymbol (oft auch Kronecker–Jacobi–Symbol oder Kroneckersymbol genannt) :

Definition A.4 (Jacobisymbol) ([Ch93, Definition 1.4.8])
Für $a, b \in \mathbb{Z}$ ist das Jacobisymbol $\left(\frac{a}{b}\right)$ wie folgt definiert

1. *Für $b = 0$ ist $\left(\frac{a}{0}\right) = 1$, wenn $a = \pm 1$, und 0 sonst.*

2. *Für $b \neq 0$ und $b = \prod p$, wobei die p nicht notwendig verschiedene Primzahlen oder -1 sind, dann ist*
$$\left(\frac{a}{b}\right) = \prod \left(\frac{a}{p}\right), \tag{A.3}$$
wobei für $p > 2$ $\left(\frac{a}{p}\right)$ das in Definition A.3 definierte Legendre Symbol und
$$\left(\frac{2}{a}\right) := \begin{cases} 0, & \text{wenn } a \text{ gerade ist} \\ (-1)^{(a^2-1)/8}, & \text{wenn } a \text{ ungerade ist} \end{cases} \tag{A.4}$$
und
$$\left(\frac{a}{-1}\right) := \begin{cases} 1, & \text{wenn } a \geq 0 \\ -1, & \text{wenn } a < 0 \end{cases} \tag{A.5}$$

ist. □

Bemerkung A.5
Mit Hilfe der in [Ch93, Theorem 1.4.9] beschriebenen Eigenschaften des Jacobisymbols $\left(\frac{a}{b}\right)$ kann man leicht zeigen, daß für ungerade $b \in \mathbb{P}$ das Jacobisymbol gleich dem Legendresymbol ist. □

Um das Jacobisymbol und damit für ungerade $b \in \mathbb{P}$ das Legendresymbol effizient berechnen zu können, benötigen wir einige Eigenschaften, deren Formulierung und Beweis in [Kn81, Übung 23, Kapitel 4.5.4] zu finden sind.

Lemma A.6 (Eigenschaften des Jacobisymbols)
Sei a, b, $c \in \mathbb{Z}$, b, $c \neq 1$, ungerade. Dann gilt

$$\left(\frac{a}{b}\right) = \left(\frac{a \bmod b}{b}\right) \tag{A.6}$$
$$\left(\frac{0}{b}\right) = 0 \tag{A.7}$$
$$\left(\frac{1}{b}\right) = 1 \tag{A.8}$$
$$\left(\frac{2}{b}\right) = (-1)^{(b^2-1)/8} \tag{A.9}$$
$$\left(\frac{c}{b}\right) = (-1)^{(b-1)(c-1)/4}\left(\frac{b}{c}\right) \tag{A.10}$$

□

Aus diesen Eigenschaften ergibt sich folgender Algorithmus zur Berechnung des Jacobisymbols.

Algorithmus A.7 (Jacobisymbol)

Jacobisymbol

EINGABE: $a, b \in \mathbb{Z}$
AUSGABE: $\left(\frac{a}{b}\right)$ gemäß Definition A.4

<u>TRIVIALFÄLLE</u>

(1)	**if** $(b=0 \wedge \|a\|=1)$ **then**	/* Definition A.4 1. erfüllt? */
(2)	**return** (1)	
(3)	**if** $(b=0 \wedge \|a\| \neq 1)$ **then**	/* Definition A.4 1. erfüllt? */
(4)	**return** (0)	
(5)	**if** $(2\|a \wedge 2\|b)$ **then**	/* nach Definition A.4 */
(6)	**return** (0)	
(7)	$jaco := 1$	/* Initialisiere Ergebnis */

<u>MACHE b UNGERADE</u>

(8)	Bestimme u, v mit $b = u \cdot 2^v$ mit u ungerade	
(9)	$jaco := jaco * (-1)^{v \cdot (a^2-1)/8}$ $b := u$	/* Nutze Multiplikativität und (A.9) */
(10)	**if** ($b < 0 \wedge a < 0$) **then**	/* Behandle Vorzeichen */
(11)	$jaco := -jaco$	/* nach Definition A.4 */
(12)	$b := \|b\|$	

<u>VERFAHREN BEENDET ?</u>

(13)	**while** (1) **do**	/* Abbruch der Schleife durch **return** */
(14)	**if** $(a=0 \wedge b>1)$ **then**	/* $\mathrm{ggT}(a,b) > 1$ */
(15)	**return** (0)	
(16)	**if** $(a=0 \wedge b=1)$ **then**	
(17)	**return** (1)	

Fortsetzung auf Seite 164

<u>TAUSCHE DIE ROLLEN VON a UND b</u>

Bestimme u, v mit $|a| = u \cdot 2^v$ mit u ungerade
$jaco := jaco * (-1)^{v\cdot(b^2-1)/8}$, $a := u$ /* Nutze (A.9) */
$jaco := jaco * (-1)^{(a-1)\cdot(b-1)/4}$ /* Nutze (A.10) */
$h := a$; $a := b$; $b := a$ /* Nutze (A.6) */
od

A.4 Primzahltests

In diesem Kapitel möchte ich einige Verfahren vorstellen, die überprüfen, ob eine gegebene Zahl zusammengesetzt ist oder nicht. Verfahren, die dies leisten, heißen Primzahltests[2] bzw. Primzahlbeweise. Der Unterschied zwischen diesen beiden Klassen von Algorithmen besteht darin, daß ein Algorithmus aus der Familie der Primzahltests im Gegensatz zu Primzahlbeweisen, nur mit hoher[3] Wahrscheinlichkeit richtig liegt, wenn er eine Zahl als Primzahl erkennt. An dieser Stelle möchte ich dennoch nur auf Primzahltests eingehen, da sie für das *Number Field Sieve* Verfahren vollkommen ausreichend sind.
Die Basis vieler Primzahltests ist

Satz A.8 (Fermatscher Satz) ([Ri85], Theorem A2.8)
Sei p eine Primzahl, $a \in \mathbb{Z}$ und $\mathrm{ggT}(a, p) = 1$, dann gilt

$$a^{p-1} \equiv 1 \pmod p \tag{A.11}$$

□

Aus dem Fermatschen Satz läßt sich direkt ein einfacher Primzahltest herleiten, denn ist $\mathrm{ggT}(a, N) = 1$ und $a^{N-1} \not\equiv 1 \pmod N$, so ist N zusammengesetzt. Leider gibt es auch zusammengesetzte Zahlen, sogenannte Pseudoprimzahlen zur Basis a, die (A.11) erfüllen. Ein einfaches Beispiel ist

Beispiel A.9 (Pseudoprimzahl zur Basis 2)
Für $N = 341 = 11 \cdot 31$ gilt

$$2^{340} \equiv 1 \pmod{341} \tag{A.12}$$

341 ist also eine Pseudoprimzahl zur Basis 2. □

[2] eigentlich müßte es richtig *Zusammengesetztheitstest*, im Englischen *compositeness test*, heißen, aber *Primzahltest* hat sich im Deutschen durchgesetzt

[3] die Qualität von *'hoher'* ist abhängig von dem jeweiligen Verfahren

Die Idee, einfach mehrere verschiedene a zu probieren, liefert auch nicht immer das gewünschte Ergebnis, denn es existieren zusammengesetzte Zahlen N, die pseudoprim zu jeder Basis a mit $\mathrm{ggT}(a, N) = 1$ sind. Diese Zahlen heißen Carmichael Zahlen. R. Alford, A. Granville und C. Pommerance haben bewiesen, daß es unendlich viele Carmichael Zahlen gibt (siehe auch [Ch93], Kapitel 8.2) und, daß es etwa x^c Carmichael Zahlen unterhalb jeder Schranke $x \in \mathbb{R}_{>0}$ gibt, wobei c nahe bei 0.1 liegt.

Beispiel A.10 (Carmichael Zahl)
Die kleinste Carmichael Zahl ist $561 = 3 \cdot 11 \cdot 17$. □

Somit liefert der Fermatsche Primzahltest kein zufriedenstellendes Verfahren. Eine Verbesserung ergibt sich aber aus dem

Satz A.11 (Euler Kriterium) ([Ri85], Theorem A3.6)
Sei p eine ungerade Primzahl, $a \in \mathbb{Z}$ und $\mathrm{ggT}(a, p) = 1$, dann gilt

$$\left(\frac{a}{p}\right) \equiv a^{\frac{p-1}{2}} \pmod{p}. \tag{A.13}$$

□

$\left(\frac{a}{p}\right)$ bezeichnet dabei das Legendre Symbol (siehe Kapitel A.3). Definition A.3, Definition A.4 und Satz A.11 liefern einen Primzahltest, der auf R. Solovay und V. Strassen ([SoSt77]) zurückgeht:
Wähle zufällig ein $a \in \{2, \ldots, N-2\}$. Ist $\mathrm{ggT}(a, N) \neq 1$, so ist N zusammengesetzt. Ist der ggT gleich 1, so berechne das Jacobi Symbol $\left(\frac{a}{b}\right)$ und vergleiche das Resultat mit $a^{\frac{N-1}{2}} \pmod{N}$. Stimmen die beiden Werte nicht überein, so ist N zusammengesetzt.
Zusammengesetzte Zahlen, für die A.13 gilt, nennt man Eulerpseudoprim zur Basis a (kurz: epsp(a)). Man kann zeigen ([Kr86], Theorem 2.28), daß für höchstens $N/2$ Werte a N epsp(a) ist. Das heißt also, wendet man das Verfahren mit k verschiedenen a an, so findet man entweder, daß N zusammengesetzt ist oder, daß N mit einer Wahrscheinlichkeit von mindestens $1 - \frac{1}{2^k}$ prim ist.
Eine bessere Erfolgsquote weist der Primzahltest von G. Miller und M. Rabin auf, dem die

Definition A.12 ([Ch93], Definition 8.2.2)
Sei N eine ungerade ganze Zahl und $a \in \mathbb{Z}$. Schreibt man $N-1$ in der Form

$$N - 1 = 2^t \cdot q \tag{A.14}$$

mit ungeradem q, so heißt N stark pseudo prim zur Basis a (kurz: spsp(a)), wenn entweder

$$a^q \equiv 1 \pmod{N} \tag{A.15}$$

oder es existiert ein $0 \le e < t$ *mit*

$$a^{2^e \cdot q} \equiv -1 \quad (mod\ N) \tag{A.16}$$

□

zugrunde liegt. Man kann leicht einsehen ([Ch93], Kapitel 8.2, Übung 1), daß jede ungerade Primzahl spsp(a) ist, für alle a mit $p \not| a$. Auf der anderen Seite läßt sich aber auch zeigen ([Kr86], Th. 2.33), daß die Anzahl der Werte $1 < a < N$, für die eine zusammengesetzte Zahl N spsp(a) ist, höchstens $N/4$ beträgt. Damit liefert der unten aufgeführte Miller-Rabin Primzahltest, angewandt mit k verschiedenen Basen a entweder, daß N zusammengesetzt ist oder, daß N mit einer Wahrscheinlichkeit von mindestens $1 - \frac{1}{4^k}$ prim ist.

Algorithmus A.13 (Miller–Rabin)

Miller–Rabin

EINGABE: $z \in \mathbb{N}_{>3}$
AUSGABE:

$$\begin{array}{ll} \text{TRUE}, & \text{wenn } z \text{ wahrscheinlich Primzahl ist} \\ \text{FALSE}, & \text{wenn } z \text{ zusammengesetzt ist} \end{array}$$

```
q := z − 1;                          /* Initialisierung */
t := 0;
while (q ist gerade) do              /* Berechnung des ungeraden
   q := q/2;                            Anteils q und des Exponenten t
   t ++;                                aus Definition A.12 */
od
Wähle a ∈{2, . . . , z − 1} zufällig /* Wähle Basis */
pot := a^q;
if (pot = 1) then
   return (TRUE)
fi
e := 0;
while (pot ≢ ±1 (mod z)∧ e ≤
         t − 2) do
   pot := pot^2 (mod z);
   e ++;
od
if (pot≠ z − 1) then
   return (FALSE)
else
   return (TRUE)
fi
```

A.5 Die Pollard ρ–Faktorisierungsmethode

In diesem Kapitel möchte ich einen Faktorisierungsalgorithmus (siehe [Ri85, Kapitel 5, Seite 174]) von J. M. Pollard ([Po75]) vorstellen, der besonders effizient für Zahlen zwischen 12 und 20 Dezimalstellen arbeitet.
Sei N eine zusammengesetzte Zahl. Konstruiert man eine Folge von ganzen Zahlen $\{x_i\}$, $i = 1, \ldots$, so besitzt diese Folge modulo N eine Periode, die im schlechtesten Falle gerade Länge N hat. Für einen Primteiler p von N hat die gleiche Folge möglicherweise eine andere Periodeinlänge. Findet man nun zwei Folgenglieder x_i und x_j, $i \neq j$ mit

$$x_i \equiv x_j \pmod{p},$$

so liefert der größte gemeinsame Teiler

$$g = \mathrm{ggT}(x_i - x_j, N) \tag{A.17}$$

einen Teiler von N, der gute Chancen hat nichttrivial zu sein. Da man p durch das Verfahren berechnen will, also während des Verfahrens nicht kennt, wird die Folge modulo N bestimmt. Am einfachsten ließe sich ein Teiler von N dadurch ermitteln, daß man für ein gerade berechnetes Folgenglied x_i alle größten gemeinsamen Teiler

$$\mathrm{ggT}(x_i - x_j, N), \quad j < i$$

berechnet. Dieses Vorgehen bedeutet aber, daß alle Folgenglieder gespeichert werden müssen, was zu großen Hauptspeicherbedarf erfordert. Deshalb verwendet man besser *Floyd's cycle–finding Algorithmus*. In diesem werden nur die Werte x_i und x_{2i} zur Berechnung des größten gemeinsamen Teilers herangezogen.

Bemerkung A.14 (Pollard's ρ-Methode in der Praxis)

1. In der Praxis am geeignetsten hat sich die Berechnung der Folge durch die Rekursionsformel
$$x_{i+1} = {x_i}^2 + 1$$
mit dem Startwert $x_1 = 2$ erwiesen.

2. Anstatt nach jedem Schritt den ggT zu bilden, kann man einige der Werte $x_i - x_j$ (z.B. 20 Differenzen) aufmultiplizieren und dann erst den ggT wie in (A.17) bilden.

3. Eine 25–prozentige Verbesserung der Laufzeit des Verfahrens in der Praxis geht auf R. P. Brent ([Br80]) zurück. Dieser schlägt vor, anstatt der Differenzen $x_{2i} - x_i$ die Differenzen $x_j - x_{2^k}$ für $3 \cdot 2^{k-1} < j \leq 2^{k+1}$ zu verwenden. Der Vorteil dieser Methode besteht darin, daß keine Werte doppelt berechnet werden müssen, wie dies in Floyd's cycle–finding Algorithmus mit allen Folgengliedern von geradem Index erfolgt.

□

Algorithmus A.15 Pollard ρ–Methode

POLLARD_RHO

EINGABE: $N \in \mathbb{N}$, zusammengesetzt
AUSGABE: n_1, $n_2 \in \mathbb{N}$ mit $N = n_1 \cdot n_2$

```
j := 1; xj := 2                      /* Initialisierung */
g := 1                               /* Initialisierung des ggT */
while (g ≠ 1) do                     /* Solange kein Teiler
                                        gefunden wurde */
   i := j, xi := xj                  /* nächste Zweierpotenz */
   for (j := i + 1; j < 3/2 i; j ++) do
      xj := xj^2 + 1                 /* unberücksichtigte Folgenglieder
                                        */
   od
   prod := 1                         /* Produkt initialisieren */
   for (; j ≤ 2 * i; j + +) do
      xj := xj^2 + 1                 /* berücksichtigte Folgenglieder */
      prod := prod · (xj − xi)       /* Differenzen aufmultiplizieren */
   od
   g = ggT(prod, N)
od
return (g, N/g)                      /* Teiler zurückliefern */
```

A.6 Irreduzibilitätstest

In diesem Kapitel werde ich die Frage behandeln, wie man überprüft, ob zu gegebenem Polynom $t(x) \in \mathbb{Z}[x]$ und einer Primzahl $p \in \mathbb{P}$ $t(x)$ modulo p irreduzibel ist oder nicht.
Die algorithmische Lösung für dieses Problem findet man in [Kn81, Kapitel 4.6.2, Übung 16]. Sie beruht auf folgendem Lemma:

Lemma A.16
Sei $t(x)$ ein Polynom vom Grad d und $p \in \mathbb{P}$. $t(x)$ ist genau dann irreduzibel modulo p, wenn gilt

1. $t(x) \mid x^{p^d} - x$
2. $\mathrm{ggT}\left(x^{p^{d/q}} - x, t(x)\right) = 1$, *für alle Primzahlen q mit $q \mid d$*

□

Für Polynome kleinen Grades ergibt sich hieraus ein einfacher Algorithmus.

Algorithmus A.17 (Irreduzibilitätstest modulo p)

IRREDTEST

EINGABE: $t(x) \in \mathbb{Z}[x]$ vom Grade d, $p \in \mathbb{P}$
AUSGABE:

$$\begin{array}{ll} \text{TRUE,} & \text{wenn } t(x) \text{ irreduzibel modulo } p \text{ ist} \\ \text{FALSE,} & \text{wenn } t(x) \text{ reduzibel modulo } p \text{ ist} \end{array}$$

<u>TEST DER BEDINGUNG $t(x) | x^{p^d} - x$</u>

(1)	$pd :=$ FASTEXPO(p, $\mathbb{Z}$, d)	/* p^d mittels Algorithmus A.1 */
(2)	$xpd :=$ FASTEXPO(x, $\mathbb{Z}[x]$, pd)	/* x^{p^d} mittels Algorithmus A.1 */
(3)	**if** ($xpd - x \neq 0$) **then**	/* Bedingung 1 aus Lemma A.16 erfüllt, wenn $xpd - x = 0$ in $\mathbb{Z}[x]/t(x)$ ist */
(4)	**return** (FALSE)	

<u>TEST DER BEDINGUNG ggT $\left(x^{p^{d/q}} - x, t(x)\right) = 1$</u>

(5)	Berechne die k Primteiler q_i von d	
(6)	**for** ($i := 1$; $i \leq k$; i ++) **do**	
(7)	$pdq := FASTEXPO(p, d/q_i)$	/* Exponentiation in $\mathbb{Z}$ */
(8)	$xpdq := FASTEXPO(x, pdq)$	/* Exponentiation in $\mathbb{Z}[x]/t(x)$ */
(9)	**if** (ggT($xpdq - x, t(x)$) $\neq 1$) **then**	/* Bedingung 2 aus Lemma A.16 erfüllt ? */
(10)	**return** (FALSE)	
(11)	**od**	
(12)	**return** (TRUE)	/* Alle Bedingungen von Lemma A.16 erfüllt */

Aus diesem Algorithmus läßt sich leicht eine probabilistische Methode herleiten, die auf Irreduzibilität in $\mathbb{Z}[x]$ testet. Gibt es nämlich mindestens eine Primzahl p, modulo der das Polynom $t(x)$ irreduzibel ist, so ist $t(x)$ auch irreduzibel in $\mathbb{Z}[x]$, denn wäre $t(x)$ über $\mathbb{Z}[x]$ zerlegbar in die beiden nichttrivialen Polynome $t_1(x)$ und $t_2(x)$, so wäre $t(x)$ modulo p auch in die Polynome $t_1(x)$ modulo p und $t_2(x)$ modulo p zerlegbar.
Für normierte irreduzible Polynome vom Grad d gilt ([BuLePo93, Kapitel 9, Seite 72]), daß $t(x)$ durchschnittlich modulo einer von d Primzahlen auch irreduzibel ist. Dies bedeutet, daß man für kleine Polynomgrade nur wenige Tests machen muß, um die Irreduzibilität zu beweisen. Liegt dagegen ein reduzibles Polynom vor, so kann

man mit diesem Verfahren nie mit Sicherheit sondern nur mit hoher Wahrscheinlichkeit sagen, daß das Polynom reduzibel ist.

Anhang B

Algorithmenverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Literaturverzeichnis

[Ad91] L. M. Adleman , *Factoring numbers using singular integers*, ACM Symp. on Theory of Computing (STOC) (1991), pp. 64 - 71

[BeLe93] Daniel J. Bernstein, A. K. Lenstra, *A general number field sieve implementaton*, Lecture Notes in Mathematics 1554, pp. 103–126, Springer Verlag, 1993

[Br80] Richard P. Brent, *An Improved Monte Carlo Factorization Algorithm* Nordisk Tidskrift för Informationsbehandling (BIT) 20 (1980) pp. 176-184

[BrRi92] Richard P. Brent, Herman J. J. te Riele, *Factorizations of* $a^n \pm 1$, $13 \leq a \leq 100$, CWI Report, Department of Numerical Mathematics NM-R9212, Juni 1992

[Br88] David M. Bressoud, *Factorization and Primality Testing* Undergraduate Texts in Mathematics, Springer verlag, 1988

[BLSTW83] J. Brillhart, D. H. Lehmer, J. L. Selfridge, B. Tuckerman, S.S. Wagstaff Jr., *Factorization of* $b^n \pm 1$, $b = 2, 3 5, 7, 10, 11, 12$ *up to high powers* American Mathematical Society, Volume 22, 1983

[BuLoZa93] J. Buchmann, J. Loho, J. Zayer, *An implementation of the general number field sieve*, Advances in Cryptology - Crypto 93 Lecture Notes in Computer Science 773, pp. 159 - 165, Spinger Verlag

[BuLePo93] J. P. Buhler, H. W. Lenstra Jr., C. Pomerance, *Factoring integers with the number field sieve*, Lecture Notes in Mathematics 1554, pp. 50 - 94, Springer Verlag, 1993

[Ch93] Henri Cohen, *A Course in Computational Algebraic Number Theorie*, Graduate Texts in Mathematics 138, Springer Verlag, 1993

[Cp93] Don Coppersmith, *Solving linear equations over GF(2): Block Lanczos algorithm*, Linear Algebra and its Applications 192 (1993), pp. 33 - 60

[Cp94] Don Coppersmith, *Solving homogeneous linear equations over GF(2) via Block Wiedemann algorithm*, Mathematics of Computation 62 (1994), no. 205, pp. 333 - 350

[Cv94] J. - M. Couveignes, *Computing a square root for the number field sieve*, Lecture Notes in Mathematics 1554, pp. 95 - 102, Springer Verlag, 1993

[De93] T. Denny, *Faktorisieren mit dem Quadratischen Sieb* Diplomarbeit, Universität des Saarlandes, 1993

[DDLM93] T. Denny, B. Dodson, A.K. Lenstra, M.S. Manasse, *On The Factorization Of RSA–120*, Proceedings Crypto 1993

[GLM94] R. A. Golliver, A. K. Lenstra, K. S. McCurley, *Lattice Sieving and trial division*, Lecture Notes in Computer Science 877, pp. 18 - 27, Springer Verlag, 1994

[Hec] E. Hecke *Algebraische Zahlen*, Chelsea Publishing Company, Bronx, New York

[Kn81] D. E. Knuth, *The Art of Computer Programming, vol. 2*, Second Edition, Addison Wesley, 1981

[Ho73] B. Hornfeck, *Algebra*, 2.Auflage, De Gruyter Lehrbuch, Berlin, New York 1973

[Kr86] E. Kranakis, *Primality and Cryptography*, Wiley-Teubner Series in Computer Science, 1986

[LaOd90] B. A. LaMacchia, A. M. Odlyzko, *Solving large sparse linear systems over finite fields* Advances in Cryptology - Proceedings of Crypto 90, volume 537 of Lecture Notes in Computer Science, pp. 72 - 82, New York, 1991, Springer Verlag

[LeTi92] H. W. Lenstra, Jr., R. Tijdeman(eds), *Computational methods in number theory* Mathematical Centre Tracts 154/155, Mathematisch Centrum, Amsterdam 1992

[Le94] A. K. Lenstra, *116–digit GNFS factorization*, e–mail im Number Theory Net vom 20.7.94

[LLL82] A. K. Lenstra, H. W. Lenstra Jr., L.Lovascz, *Factoring polynomials with rational coefficients*, Math. Ann. 261 (1982), 515-534

[LLMP90] A. K. Lenstra, H. W. Lenstra Jr., M. S. Manasse, J. M. Pollard, *The number field sieve*, Abstract: Proc. 22nd Ann. ACM Symp. on Theory of Computing (STOC) (1990), 564-572

[LeMa] A. K. Lenstra, M. Manasse, *Factoring with two large primes*, Math. Comp. , to appear

[Pa93] T. Papanikolaou, *libF - eine lange Gleitpunktarithmetik*, Diplomarbeit, Universität des Saarlandes, 1993

[Po93] M. Pohst, *Computational Algebraic Number Theorie*, DMV Seminar, Band 21 Birkhäuser Verlag 1993

[PoZa89] M. Pohst & H. Zassenhaus, *Algorithmic algebraic number theorie*, Cambridge University Press 1989

[Po88] J. M. Pollard, *Factoring with cubic integers*, Manuskript, August 1988, Lecture Notes in Mathematics 1554, pp. 4 - 10, Springer Verlag, 1993

[Po91] J. M. Pollard, *The Lattice Sieve*, Manuskript, September 1991, Lecture Notes in Mathematics 1554, pp. 43 - 49, Springer Verlag, 1993

[Po75] J. M. Pollard, *A Monte Carlo Method for Factorization*, Nordisk Tidskrift för Informationsbehandling (BIT) 15 (1975) pp. 331-334

[Ri85] H. Riesel, *Prime Numbers and Computer Methods for Factorization*, Second Printing, Birkkäuser Verlag, 1985

[Ro92] R. Roth, *LiPS, ein System für verteilte Anwendungen* Diplomarbeit, Universität des Saarlandes, 1992

[Se91] T. Setz, *Integration einer Linda–orientierten Laufzeitumgebung in LiPS* Diplomarbeit, Universität des Saarlandes, 1991

[Sh72] D. Shanks, *Five Number-Theoretic Algorithms* , Proc. Second Manitoba Conference On Numerical Math., 1972,

[SoSt77] R. Solovay, V. Strassen, *A fast Monte-Carlo test for primality*, SIAM J. Comput. 6 (1977); erratum ibid. 7 (1978)

[StTa87] I. N. Stewart, D. O. Tall, *Algebraic number theory*, second edition, Chapman and Hall, 1987

[Va91] B. Vallée, *Gauß' algorithm revisited*, Journal of Algorithms, pp. 123 - 131, 1991.

[Wa76] P. S. Wang, *Factoring multivariate polynomias over algebraic number fields*, Math. Comp. 30 (1976), pp. 324 - 336

[Wr76] P. J. Weinberger, L. P. Rothschild, *Factoring polynomials over algebraic number fields*, ACM, Trans. Math. Software 2, (1976), pp. 335 - 350

[Ws76] E. Weiss, *Algebraic number theory*, McGraw-Hill, New York, 1963, reprinted, Chelsea, New York, 1976

[Za91] J.Zayer, *die Theorie des number field sieve*, Diplomarbeit, Universität des Saarlandes, 1991