Faktorisieren mit dem Quadratischen Sieb

Diplomarbeit

von

Thomas Friedrich Denny

nach einem Thema von Prof. Dr. Buchmann,
am Fachbereich 14 – Informatik –
der Universität des Saarlandes

April 1993

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

Saarbrücken,

Mein Dank gebührt Herrn Prof. Dr. Buchmann für die Vergabe dieses interessanten Themas sowie der motivierenden Betreuung während der Ausarbeitung dieser Arbeit.

Mein Dank gilt auch Arjen K. Lenstra, der sich nicht nur per Email als kompetenter und hilfsbereiter Ansprechpartner erwiesen hat.

Des weiteren bedanke ich mich bei Volker Müller und Jörg Zayer für das Korrekturlesen der Arbeit.

Außerdem bedanke ich mich bei Jutta Bartholomes, Jürgen Loho, Sachar Paulus und Jörg Zayer für viele anregende Diskussionen und für die angenehme Arbeitsatmosphäre.

Franz-Dieter Berger danke ich für produktive Tips und die hervorragende Zusammenarbeit, die auch zu manch' lustiger Nacht vor dem Terminal führte.

Mein besonderer Dank gilt meinen Eltern und meinem Geschwistern, die mein Studium ermöglichten und mich nach besten Kräften unterstützt haben.

Inhaltsverzeichnis

Tabellenverzeichnis

Abbildungsverzeichnis

Einleitung

Diese Arbeit beschäftigt sich mit der Zerlegung großer natürlichen Zahlen in ihre Primfaktoren. Die Berechnung dieser Darstellung heißt Faktorisierung. Die Faszination dieses Problems, das sich auch jedem Nicht-Mathematiker leicht erklären läßt, kann an der Tatsache, daß sich schon seit Jahrhunderten die größten Mathematiker mit diesem Problem beschäftigt haben, erahnt werden. Eine gute Zusammenfassung der Geschichte der Faktorisierung findet sich bei [Leh]. Die Faktorisierung spielt in vielen Bereichen der Mathematik eine große Rolle. So wird z.B. bei der Lösung bestimmter zahlentheoretischer Probleme die Primfaktorzerlegung benötigt, und die Faktorisierung dient der Erforschung der natürlichen Zahlen.

Durch die zunehmende Bedeutung der Kryptographie in den letzten Jahrzehnten wuchs auch die praktische Bedeutung der Faktorisierung. Der stark wachsende Einsatz von Computern zur Speicherung und Verarbeitung sensibler Daten und deren Übertragung über öffentlich zugängliche Netze haben zum verstärkten Einsatz von Kryptosystemen geführt. Die Sicherheit von Krytosystemen ist eng mit dem Problem der Faktorisierung verknüpft. Die zunehmende Rechenleistung der Computer und immer bessere Verfahren greifen die Sicherheit dieser Systeme an. Deshalb ist es von Interesse, die Möglichkeiten der Faktorisierung abzuschätzen und so zur Sicherheit bestehender Krytosysteme beizutragen.

Generell teilt man Faktorisierungsalgorithmen in zwei Klassen ein :

1. Allgemeine Faktorisierungsverfahren, deren Laufzeit im wesentlichen von der Größe der zu faktorisierenden Zahl abhängt.

2. Spezielle Faktorisierungsverfahren, deren Laufzeit im wesentlichen von besonderen Eigenschaften der Zahl abhängt.

Der von mir beschriebene und implementierte Faktorisierungsalgorithmus 'Das Quadratische Sieb' (QS) gehört zu den allgemeinen Faktorisierungsverfahren. Seine Laufzeit ist nur von der Größe der zu faktorisierenden Zahl abhängig und nicht von der Größe der Primfaktoren oder einer speziellen Form der Zahl. Die größte Zahl, die bisher weltweit mit dem Quadratischen Sieb faktorisiert wurde, hatte 116 Dezimalstellen. Dazu wurde eine Rechenzeit von etwa 400 MIPS-Jahren benötigt, was etwa $1.26 \cdot 10^{12}$ Integer-Operationen entspricht. Dies stellt auch den bisherigen Weltrekord eines allgemeinen Faktorisierungsalgorithmus dar. Geht man von der realistischen Annahme der Verdoppelung der Rechenzeit bei einer Erhöhung der Dezimalstellen der zu faktorisierenden Zahl um drei Stellen aus, kann man nachvollziehen, daß es im Moment nicht möglich ist, eine 200-stellige Dezimalzahl mit einem bekannten allgemeinen Faktorisierungsalgorithmus zu zerlegen. Im Moment (April 1993) läuft ein neuer Weltrekordversuch, an dem das von mir implementierte Programm beteiligt ist. Es handelt sich dabei um eine 120-stellige Zahl (RSA120) aus dem RSA-Wettbewerb (vgl. Kapitel 6). An diesem Versuch sind außerdem Bruce Dodson, Arjen K. Lenstra und Mark Manasse beteiligt. Die Faktorisierung dürfte Mitte dieses Jahres abgeschlossen sein.

Als sehr erfolgreich erwies sich auch die Zusammenarbeit meines QS-Programms mit dem von Franz-Dieter Berger implementierten ECM(Elliptic Curve Method)-Programm (vgl. [Ber]). ECM ist ein spezielles Faktorisierungsverfahren, das sehr schnell kleinere Primfaktoren einer Zahl bestimmt. Während dieser Zusammenarbeit wurden bisher von meinem Programm 31 Zahlen faktorisiert, die zwischen 75 und 104 Dezimalstellen haben. Dabei hat der Großteil dieser Zahlen ungefähr 98 Dezimalstellen. Mit meiner QS-Version wurden außerdem noch fünf 84-stellige Zahlen aus dem erweiterten Cunningham Projekt ([BreRie]) faktorisiert.

Meine Arbeit gliedert sich wie folgt :

Kapitel 1:
Hier werden die benutzten Bezeichner, Definitionen und Sätze vorgestellt. Außerdem erläutere ich die Darstellung der Algorithmen.

Kapitel 2:
Die Grundidee einer großen Klasse von Faktorisierungsalgorithmen und die prinzipielle Vorgehensweise des Quadratischen Siebes bilden das zweite Kapitel.

Kapitel 3:
In Kapitel 3 wird die Einteilung des Algorithmus in Phasen und Module vorgenommen. Ausgehend von der Grundversion wird eine erste Modifikation des Grundalgorithmus vorgestellt. Bei der Beschreibung der einzelnen Teilmodule wird jeweils zuerst der theoretische Hintergrund erklärt, bevor auf einige wichtige praktische Aspekte eingegangen wird.

Kapitel 4:
In diesem Kapitel werden Erweiterungen des Basisalgorithmus vorgestellt und deren Auswirkungen auf die einzelnen Phasen erläutert. Die hier vorgestellten Erweiterungen haben die praktischen Einsatzmöglichkeiten des Algorithmus stark verbessert. Es werden praktische Erfahrungen mit den Erweiterungen vorgestellt, und sofern dies möglich, ist eine genaue Analyse ihres Verhaltens gegeben.

Kapitel 5:

Die Programmbeschreibung schließt sich in Kapitel 5 an. Hier werden die Benutzung und einige Eigenschaften des Programms beschrieben.

Kapitel 6:

In letzten Kapitel wird die erfolgreiche Teilnahme an einem internationalen Faktorisierungswettbewerb der Firma “RSA Data Security” beschrieben. Außer dem Wettbewerb, an dem Franz-Dieter Berger und ich teilnehmen, beschreibe ich das Zusammenarbeiten der beiden Algorithmen (ECM und QS).

Kapitel 1

Grundlagen

Im einführenden Kapitel werde ich die benutzten Bezeichnungen vorstellen, die Formulierung und Darstellung der Algorithmen erklären und die benutzte O-Notation besprechen. Außerdem erläutere ich benutzte Sätze und Definitionen.

1.1 Bezeichnungen

Zur Bezeichnung der Zahlbereiche verwende ich die folgenden Symbole:

$\mathbb{N}$ Menge der natürlichen Zahlen (ohne Null),

$\mathbb{P}$ Menge der Primzahlen,

$\mathbb{Z}$ Menge der ganzen Zahlen,

$\mathbb{Q}$ Menge der rationalen Zahlen.

$\mathbb{R}$ Menge der reellen Zahlen.

Die folgenden Bezeichnungen werden später vorausgesetzt :

- Der Begriff "lange Zahl" wird für eine ganze Zahl verwendet, die nicht im normalen Datentyp Integer abgespeichert werden kann. Sie wird in einer speziellen Datenstruktur, der verwendeten langen Arithmetik ([Dül]) abgespeichert. Kann eine ganze Zahl im Datentyp Integer abgespeichert werden, so wird sie als "kurze Zahl" bezeichnet.

- Für eine endliche Menge M bezeichnet $|M|$ die Anzahl der Elemente dieser Menge oder auch die Mächtigkeit der Menge M.

- Für $x \in \mathbb{R}$ bezeichnet $\lfloor x \rfloor$ die größte ganze Zahl a mit $a \leq x$ und $\lceil x \rceil$ die kleinste ganze Zahl b mit $b \geq x$.
- Für $a, b \in \mathbb{Z}$ schreiben wir $a|b$ (bzw. $a \nmid b$) wenn a ein (bzw. kein) Teiler von b ist.
- Für $a, b \in \mathbb{Z}$ und $m \in \mathbb{N}$ schreiben wir $a \equiv b \bmod m$, wenn gilt $m|(a-b)$; wir bezeichnen mit $a \bmod m$ den kleinsten positiven Rest der ganzzahligen Division von a durch m.

1.2 Benutzte Sätze und Definitionen

Satz 1.1 (Fundamentalsatz der Arithmetik)
Jede von Eins verschiedene natürliche Zahl ist als Produkt endlich vieler Primzahlen darstellbar. Diese Darstellung ist bis auf die Reihenfolge der Faktoren eindeutig.

Beweis: [Rie, S.263] ■

Definition 1.2 (O-Notation)
Für zwei Funktionen $f, g : \mathbb{N} \to \mathbb{R}$ sagen wir :

$$f = O(g) \iff \underset{n_0, c \in \mathbb{N}}{\exists} \; \underset{n \geq n_0}{\forall} \; f(n) \leq c \cdot g(n)$$

$$f = o(g) \iff \lim_{n \to \infty} \frac{f(n)}{g(n)} = 0$$

Definition 1.3 (L_N-Funktion)
Für $0 \leq \nu,\ \gamma \leq 1$ und $N \in \mathbb{N}$

$$L_N[\nu, \gamma] := \exp\Big((\gamma + o(1)) \cdot (\log N)^\nu \cdot (\log\log N)^{1-\nu}\Big).$$

Definition 1.4 (ρ-Funktion)
Für ein festes $L \in \mathbb{N}$, eine zufällig gewählte Zahl $Y \in [1 \ldots L]$ und $\alpha \geq 1$ mit

$$p_i > p_{i+1}, \quad p_i \in \mathbb{P} \text{ und } p_i \,|\, Y$$

ist $\rho(\alpha)$ definiert als die Wahrscheinlichkeit, daß der größte Primfaktor p_1 von Y kleiner als $L^{\frac{1}{\alpha}}$ ist.

$$\rho(\alpha) := \lim_{L \to \infty} Prob(p_1 < L^{\frac{1}{\alpha}})$$

Die ρ-Funktion erfüllt die Differentialgleichung

$$\alpha\rho'(\alpha) + \rho(\alpha - 1) = 0$$

und kann durch numerische Integration aus

$$\rho(\alpha) = \begin{cases} 1, & \text{für } 0 \leq \alpha \leq 1 \\ \frac{1}{\alpha} \int\limits_{\alpha-1}^{\alpha} \rho(t)dt, & \text{für } \alpha > 1 \end{cases}$$

berechnet werden. ([KnuPar, S.333])

Für $5 < \alpha < 11$ kann folgende Approximation benutzt werden ([Pom, S.34]) :

$$\rho(\alpha) \approx \exp\left(-\alpha \cdot \left(\log(\alpha) + 0.56 - \frac{1}{\log(\alpha)}\right)\right) \tag{1.1}$$

Die Funktionswerte von $\rho(x)$ für $x \in [1, 12)$ mit Schrittweite 0.01 befinden sich im Anhang B. Die Funktionswerte wurden mit numerischer Integration mit Schrittweite 0.001 berechnet.

Definition 1.5 (μ-Funktion)
Für ein festes $L \in \mathbb{N}$, eine zufällig gewählte Zahl $Y \in [1 \dots L]$ und $\alpha - 1 \geq \beta \geq 1$ mit

$$p_i > p_{i+1}, \quad p_i \in \mathbb{P} \textit{ und } \; p_i \,|\, Y$$

ist $\mu(\alpha, \beta)$ definiert als die Wahrscheinlichkeit, daß der zweitgrößte Primfaktor p_2 von Y kleiner als $L^{\frac{1}{\alpha}}$ ist und der größte Primfaktor p_1 von Y kleiner als $L^{\frac{\beta}{\alpha}}$.

$$\mu(\alpha, \beta) := \lim_{L \to \infty} Prob(p_2 < L^{\frac{1}{\alpha}},\ p_1 < L^{\frac{\beta}{\alpha}})$$

Der Wert der μ-Funktion kann mit Hilfe der ρ-Funktion (vgl. Definition 1.4) und

$$\mu(\alpha, \beta) = \rho(\alpha) + \int\limits_{\alpha-\beta}^{\alpha-1} \frac{\rho(t)}{\alpha - t} dt$$

durch numerische Integration berechnet werden.([Brent, S.152])

Satz 1.6
Sei f eine Funktion. Dann kann die Summe aller Funktionswerte von $f(p)$ mit $p \in \mathbb{P}$ und $p \leq x$ folgendermaßen abgeschätzt werden:

$$\sum_{p \leq x} f(p) \approx \int_2^x \frac{f(y)}{\ln(y)} dy$$

Beweis: [RosSch] ∎

Definition 1.7 (π-Funktion)
Mit $\pi(x)$ wird die Anzahl der Primzahlen bezeichnet, die kleiner als x sind.

$$\pi : \mathbb{R}_{>0} \to \mathbb{N}_0$$

$$\pi(x) := \#\{p \in \mathbb{P} : p \leq x\}$$

Für die Werte der π-Funktion gelten folgende Abschätzungen : ([Rie, S.45])

$$\pi(x) \approx \frac{x}{\ln(x) - 1.08366} \qquad \text{(Legendre)}$$

$$\pi(x) \approx \int_2^x \frac{dx}{\ln(x)} \qquad \text{(Gauss)}$$

Definition 1.8 (Legendre-Symbol)
Für eine ungerade Primzahl $p \in \mathbb{P}$ und eine ganze Zahl $a \in \mathbb{Z}$ ist das Legendre-Symbol folgendermaßen definiert :

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & , \textit{falls } \mathrm{ggT}(a,p) = 1 \textit{ und } a \textit{ ein Quadrat modulo } p \textit{ ist} \\ -1 & , \textit{falls } \mathrm{ggT}(a,p) = 1 \textit{ und } a \textit{ kein Quadrat modulo } p \textit{ ist} \\ 0 & , \textit{falls } p|a \end{cases}$$

Falls $\left(\frac{a}{p}\right) = 1$ ist, so nennt man a einen quadratischen Rest modulo p .

Zur Berechnung des Legendre-Symbols verwende ich das Euler-Kriterium.

Satz 1.9 (Euler-Kriterium)
Sei p eine ungerade Primzahl und a eine positive ganze Zahl mit $\mathrm{ggT}(a,p) = 1$. Dann ist :

$$a^{(p-1)/2} \equiv \left(\frac{a}{p}\right) \bmod p$$

Beweis: Siehe z.B. [Kob, S.42] ■

Definition 1.10 (Partitionszahl)
Für $n \in \mathbb{N}$ ist die Partitionszahl $p(n)$ definiert als die Anzahl der (bis auf Vertauschung) verschiedenen Möglichkeiten, die Zahl n als Summe natürlicher Zahlen darzustellen.

Beispiel:
$p(5) = 7$, denn es gibt folgende 7 Möglichkeiten:

$$\begin{array}{ll}
(1) & 5 = 5 \\
(2) & 5 = 4 + 1 \\
(3) & 5 = 3 + 2 \\
(4) & 5 = 3 + 1 + 1 \\
(5) & 5 = 2 + 2 + 1 \\
(6) & 5 = 2 + 1 + 1 + 1 \\
(7) & 5 = 1 + 1 + 1 + 1 + 1
\end{array}$$

1.3 Darstellung der Algorithmen

Zur Darstellung der Algorithmen verwende ich eine modifizierte Form der Nassi-Shneidermann-Diagramme, die in [NasShn] ausführlich beschrieben werden. In diesen Diagrammen werden Algorithmen als Aneinanderreihung sogenannter Strukturblöcke dargestellt. Jeder Strukturblock erhält eine fortlaufende Nummer. Bei der Beschreibung der Algorithmen wird diese Nummer bei Verweisen benutzt.
Der einfachste dieser Strukturblöcke ist der Anweisungsblock:

Anweisung	(1)

Unter einer Anweisung versteht man dabei eine Zuweisung oder einen Prozeduraufruf. Für den Rest dieses Abschnittes verwende ich den Anweisungsblock stellvertretend für einen oder mehrere solcher Anweisungsblöcke.

Zur Darstellung von Schleifen stehen folgende Strukturblöcke zur Verfügung:

1. Zählschleifen

FOR $i \leftarrow$ Anfangswert TO Endwert		(1)
	Anweisung	(2)

Es handelt sich hierbei um eine Zählschleife, in der aufwärts gezählt wird. Die Zählvariable i wird mit dem *Anfangswert* initialisiert und dann bis zum *Endwert* hochgezählt. Dabei sind *Anfangswert* und *Endwert* Konstanten oder Variablen. Ist der *Anfangswert* größer als der *Endwert*, so nennt man die Zählschleife **abweisend**. In Zählschleifen, in denen abwärts gezählt wird, ist in obigem Strukturblock TO durch DOWNTO ersetzt. Diese werden als abweisend bezeichnet, wenn der *Anfangswert* kleiner als der *Endwert* ist. Dieses Zählschleifenkonstrukt wird nur dann verwendet, wenn innerhalb des Schleifenrumpfes weder die Zählvariable, der *Anfangswert* noch der *Endwert* (falls es sich dabei um Variablen handelt) verändert werden.

2. Schleifen, deren Fortsetzung bzw. Beendigung von einer oder mehreren Bedingungen abhängt. Man unterscheidet dabei WHILE-Schleifen und REPEAT-Schleifen. Der Unterschied dieser beiden Konstrukte besteht darin, daß der Rumpf der WHILE-Schleife gegebenenfalls überhaupt nicht ausgeführt wird, während der Rumpf der UNTIL-Schleife mindestens einmal ausgeführt wird, da die Abbruchbedingung erst am Ende jedes Schleifendurchlaufes überprüft wird.

WHILE Bedingung		(1)
	Anweisung	(2)

	Anweisung	(1)
UNTIL Bedingung		(2)

Als Kontrollstrukturen stehen uns zur Verfügung:

IF Bedingung		(1)
THEN	Anweisung1	(2)
ELSE	Anweisung2	(3)

Dabei wird die Bedingung ausgewertet und liefert als Wert entweder TRUE oder FALSE. Liefert die Auswertung TRUE, so wird Anweisung1 ausgeführt; bei FALSE wird Anweisung2 ausgeführt. Danach wird jeweils die Abarbeitung der Konstrukte beim nächsten Konstrukt fortgesetzt. Ist der Anweisungsteil 2 leer, so kann der ELSE-Fall auch weggelassen werden.

IF Bedingung		(1)
THEN	Anweisung	(2)

Neben den bisher erläuterten Strukturblöcken wird die Anweisung RETURN(z) verwendet, mit deren Hilfe eine Funktion (eine Prozedur) vorzeitig beendet wird und $z \in \mathbb{Z}$ als Ergebnis zurückgeliefert wird.

Zur Kennzeichnung von Kommentaren innerhalb der Algorithmen verwende ich die in der Programmiersprache "C" übliche Notation.

Anweisung /* Kommentar */	(1)

Kapitel 2

Prinzipielle Vorgehensweise des Quadratischen Siebs (QS)

In diesem Kapitel werde ich zuerst die Grundidee einer großen Klasse von Faktorisierungsalgorithmen beschreiben. Danach erläutere ich die prinzipielle Vorgehensweise des Quadratischen Siebes und eine grobe Einteilung in Phasen.

2.1 Grundidee

Viele Faktorisierungsalgorithmen basieren auf der Konstruktion einer quadratischen Kongruenz. Diese Strategie geht auf Fermat zurück und wurde dann z.B. vom M. Kraitchik (1929), R.E. Powers und H.D. Lehmer (1931) aufgegriffen und immer wieder verfeinert. (Siehe z.B. [Bre, S.63] , [Rie, S.146])

Sei $N \in \mathbb{Z} \quad (N \notin \mathbb{P})$ die zu faktorisierende Zahl.

Findet man eine Kongruenz

$$X^2 \equiv Y^2 \bmod N \tag{2.1}$$

mit der Eigenschaft

$$X \not\equiv \pm Y \bmod N,$$

dann gilt :

$$\begin{aligned} & N | X^2 - Y^2 = (X+Y) \cdot (X-Y) \\ & N \nmid (X-Y) \quad \text{und} \quad N \nmid (X+Y) \\ \Longrightarrow \quad & \mathrm{ggT}(X+Y, N) \quad \text{und} \quad \mathrm{ggT}(X-Y, N) \quad \text{sind echte Teiler von} \quad N\ . \end{aligned}$$

Die Algorithmen, die zu der obengenannten Familie gehören, unterscheiden sich nur in der Art und Weise, wie eine solche Kongruenz (2.1) gefunden bzw. konstruiert wird.

2.2 Vorgehensweise beim QS

Beim Quadratischen Sieb wird diese Kongruenz durch Kombination von Kongruenzen r_i der Form

$$X_i^2 \equiv Y_i \bmod N \tag{2.2}$$

erzeugt. Kongruenzen dieser Form werden im folgenden als *Relationen* (*relations*) bezeichnet. Man sammelt viele dieser *Relationen* und bestimmt davon eine Teilmenge S, so daß gilt :

$$\prod_{r_i \in S} Y_i \equiv Y^2 \bmod N \ .$$

Zur Bestimmung dieser Teilmenge benötigt man die Primfaktorzerlegung der Y_i. Ich werde später noch genauer auf die Ermittelung der Menge S eingehen, mit deren Hilfe man eine Kongruenz der Form (2.1) konstruieren kann. Es gilt

$$X^2 \equiv \prod_{r_i \in S} X_i^2 \equiv \prod_{r_i \in S} Y_i \equiv Y^2 \bmod N \ .$$

Aufgrund dieser prinzipiellen Vorgehensweise teilt man den Algorithmus in 2 Phasen ein. In der ersten Phase werden Relationen gesammelt und in der zweite Phase werden die gefundenen Relationen zu einer quadratischen Kongruenz kombiniert.

2.2.1 1. Phase : Sammeln der Relationen

Zum Erzeugen von Relationen der Form (2.2) wählt man sich ein Polynom $Q(z)$ zweiten Grades, dessen Funktionswerte Quadrate modulo N sind :

$$\bigvee_{z \in \mathbb{Z}} \ \exists_{H \in \mathbb{Z}} \quad Q(z) \equiv H^2 \bmod N.$$

Die geschickte Wahl dieses Polynoms spielt eine große Rolle für die Laufzeit des Algorithmus und ich werde später noch genauer auf die Wahl "guter" Polynome eingehen. Die erste Implementierung des QS verwendete das Polynom :

$$Q(z) := (z + \lfloor \sqrt{N} \rfloor)^2 - N$$

Die Größenordnung der Funktionswerte ist wichtig, da man ihre Zerlegung in Primfaktoren benötigt. Für die Größenordnung der Funktionswerte gilt:

$$Q(z) < 2\sqrt{N}\,|z| + z^2 \quad \text{und} \quad Q(z) > 0 \quad \text{falls } z \neq 0 \tag{2.3}$$

also

$$Q(z) = O\left(\sqrt{N}\right) \quad (\text{ für } |z| << \sqrt{N}) \ .$$

Der Vorteil dieses Polynoms liegt darin, daß man nun nur noch Zahlen in der Größenordnung von $O\left(\sqrt{N}\right)$ zu faktorisieren braucht. Um die Funktionswerte zu zerlegen, wählt man sich eine feste Menge von Primzahlen, Faktorbasis (FB) genannt, und

eine Schranke $M \in \mathbb{Z}$. Man versucht mit Hilfe eines "Siebes" systematisch Funktionswerte des Polynoms $Q(z)$ (mit $z \in [-M, M]$) über dieser Faktorbasis zu zerlegen. Diese Methode gab dem Algorithmus den Namen.
Man sammelt die Zerlegungen, die komplett über der Faktorbasis zerfallen und erhält Relationen der Form

$$Q(z_j) = \prod_{p_i \in FB} p_i^{e_{ij}} \; (= Y_j) \equiv X_j^2 \bmod N, \qquad \text{mit } e_{ij} \in \mathbb{N} \cup \{0\} \; . \tag{2.4}$$

2.2.2 2. Phase : Kombinieren der Relationen

Sei nun $F = |FB|$ die Anzahl der Elemente in der Faktorbasis und l die Anzahl der gefundenen Relationen der Form (2.4).

Dann ist $v_j = \begin{pmatrix} e_{1,j} \\ \vdots \\ e_{F,j} \end{pmatrix}$ der zur j-ten Relation zugehörige Exponentenvektor.

Um eine Teilmenge der Relationen zu finden, die kombiniert ein Quadrat auf beiden Seiten der Relation liefert, löst man das folgende Gleichungssystem über $\mathbb{F}_2$.

$$\begin{pmatrix} e_{1,1} & \cdots & e_{1,l} \\ \vdots & & \vdots \\ e_{F,1} & \cdots & e_{F,l} \end{pmatrix} \cdot \begin{pmatrix} \lambda_1 \\ \vdots \\ \lambda_l \end{pmatrix} \equiv \begin{pmatrix} 0 \\ \vdots \\ 0 \end{pmatrix} \bmod 2 \tag{2.5}$$

Um immer eine nichttriviale Lösung zu finden, muß man $l > F$ erfolgreiche Zerlegungen sammeln. Dann ist das obige Gleichungssystem überbestimmt und hat immer eine nichttriviale Lösung. Hat man eine Lösung gefunden, so übernehmen die λ_j eine "Schalterfunktion". Ist $\lambda_j = 1$ so wird die j-te Relation zur Erzeugung der gesuchten quadratischen Kongruenz benötigt. Es gilt :

$$\prod_{j=1}^{l} \left(\prod_{p_i \in FB} p_i^{e_{i,j}} \right)^{\lambda_j} \equiv Y^2 \bmod N \tag{2.6}$$

und

$$\prod_{j=1}^{l} \left(X_j^2 \right)^{\lambda_j} \equiv X^2 \bmod N \tag{2.7}$$

und nach Konstruktion

$$X^2 \equiv Y^2 \bmod N \; .$$

Man hat eine Kongruenz der Form (2.1) gefunden, berechnet sich X und Y und kann testen, ob der $\mathrm{ggT}(X+Y, N)$ und der $\mathrm{ggT}(X-Y, N)$ echte Teiler von N sind. Ist dies nicht der Fall, so bestimmt man eine weitere Lösung des Gleichungssystems (2.5). (Dafür muß man gegebenfalls weitere Relationen der Form (2.4) bestimmen.)

Kapitel 3

Der Basis-Algorithmus

In diesem Kapitel werde ich zuerst eine Modifizierung des Grundalgorithmus vorstellen. Dann gehe ich genauer auf die Einteilung des Algorithmus in die einzelnen Phasen und Module ein. Ich werde jeweils zuerst die theoretischen Grundlagen der Module vorstellen und dann auf die praktischen Aspekte bei deren Implementierung eingehen.

3.1 Die multipolynomielle Version des QS

In der Grundversion des Quadratischen Siebes wählt man sich ein Polynom und vergrößert sukzessive den Siebbereich, bis man genügend Relationen (mehr als Faktorbasiselemente) gefunden hat. Die Wahrscheinlichkeit, daß ein Funktionswert komplett über der Faktorbais zerfällt, kann mit der ρ-Funktion (vgl. Definition 1.4) berechnet werden. Dabei wird L in der Größenordnung der Funktionswerte (vgl. (2.3)) gewählt und danach α bestimmt, so daß

$$L^{\frac{1}{\alpha}} = p_{max} := \max\{p \in FB\}$$

Das Vergrößern des Siebintervalls (und damit der zu zerlegenden Funktionswerte (vgl. Kapitel 2.2.1) führt zu einer starken Verkleinerung dieser Wahrscheinlichkeit. Zur Verdeutlichung dieses Effektes, betrachtet man die fiktive Faktorisierung einer Zahl $N \approx 10^{100}$ mit der Grundversion des Quadratischen Siebes. Dies führt, bei einer realistischen Faktorbasisgröße ($|FB| \approx 10^5$) zu dem Siebintervall $[1,\ 5 \cdot 10^{15}]$. Die Tabelle 3.1 (aus [Pom, S.34]) zeigt für verschiedene Teilintervalle des gesamten Siebarrays die Wahrscheinlichkeit und die Anzahl der zu erwartenden kompletten Zerlegungen. Die Wahrscheinlichkeit, einen Funktionswert, der komplett über der Faktorbasis zerfällt, zu finden, ist im Durchschnitt im gesamten Intervall $\approx 2 \cdot 10^{-11}$. Im Intervall $[1, 10^8]$ ist diese Wahrscheinlichkeit 50-mal höher. Diese Beobachtungen führten dazu, daß man versuchte, das Siebintervall klein zu halten. Dies wird durch die Verwendung mehrerer Polynome, die die an das Originalpolynom gestellten

Teilsiebintervall	Prob[Zerlegung]	erwartete Anzahl
$[1, 10^9]$	10^{-9}	1
$[10^9, 10^{10}]$	$4.5 \cdot 10^{-10}$	4
$[10^{10}, 10^{11}]$	$2.6 \cdot 10^{-10}$	23
$[10^{11}, 10^{12}]$	$1.5 \cdot 10^{-10}$	131
$[10^{12}, 10^{13}]$	$8.4 \cdot 10^{-11}$	753
$[10^{13}, 10^{14}]$	$4.5 \cdot 10^{-11}$	4050
$[10^{14}, 10^{15}]$	$2.6 \cdot 10^{-11}$	23100
$[10^{14}, 10^{15}]$	$1.6 \cdot 10^{-11}$	65000

Tabelle 3.1: Wahrscheinlichkeit von kompletten Zerlegungen in Teilsiebintervallen

Bedingungen erfüllen, erreicht. Im Gegensatz zur Grundversion des Quadratischen Siebes wählt man eine feste Siebintervallänge und variiert die Anzahl der benutzten Polynome. Diese Idee geht auf Peter Montgomery zurück.

3.2 Einteilung in Phasen

Die vorgestellte Modifizierung hat auch Einfluß auf die Einteilung des Algorithmus in Phasen. Betrachtet man die Funktionalität des modifizierten Algorithmus, so ergibt sich die folgende feinere Einteilung in Phasen bzw. in Module. Dabei gehört Schritt (1) - (5) zur früheren Grobphase *Sammeln der Relationen* und Schritt (6) stellt die *Kombination der Relationen* dar.

Aufstellen der Faktorbasis	(1)
WHILE (nicht genügend Relationen der Form (2.4) gefunden)	(2)
Wähle ein neues Polynom	(3)
Führe die Siebphase durch	(4)
Werte die Siebphase aus	(5)
Löse das lineare Gleichungssystem über $\mathbb{F}_2$	(6)

In der Praxis wird das lineare Gleichungssystem (Schritt 6) erst dann gelöst, wenn $k > F + c$ ist (wobei $c \in \mathbb{N}$ eine kleine Konstante ist). Sobald $c > 10$ ist, existieren erfahrungsgemäß mehrere verschiedene Lösungen, mit deren Hilfe man alle nicht trivialen Teiler von N findet.

3.3 Aufstellen der Faktorbasis (FB)

In der Faktorbasis sind die ersten F Primzahlen zusammengefaßt, die mögliche Teiler eines Funktionswertes der verwendeten Polynome sind. Zur Behandlung negativer Funktionswerte wird zusätzlich auch die -1 aufgenommen. Um ein Kriterium für die Auswahl der Faktorbasiselemente zu finden, untersucht man die Eigenschaften der Primzahlen, die einen Funktionswert $Q(z)$ teilen.
Aus

$$\begin{aligned} & p \mid Q(z) \\ \iff & (z + \lfloor \sqrt{N} \rfloor)^2 - N \equiv 0 \bmod p \\ \iff & N \equiv (z + \lfloor \sqrt{N} \rfloor)^2 \bmod p \end{aligned}$$

erhält man, daß N ein Quadrat modulo p ist.

Um festzustellen, ob eine Primzahl $p > 2$ in die Faktorbasis aufgenommen werden soll, ermittelt man deshalb den Wert des Legendre-Symbols (vgl. Definition 1.8). Ist dieser 1, so ist N ein Quadrat modulo p. Das Legendre-Symbol kann nur die Werte 1 oder -1 annehmen, da vor dem Programmstart durch Probedivisionen oder spezielle Faktorisierungsalgorithmen (z.B. ECM vgl. [Ber]) sichergestellt wird, daß N keine kleinen Faktoren hat und somit

$$\forall_{p \in \mathbb{P},\ p \leq 10^7} \quad \mathrm{ggT}(N, p) = 1 \quad \text{gilt.}$$

Die Faktorbasis hat dann (in Abhängigkeit von N und einer zu wählenden Schranke B) folgendes Aussehen:

$$FB_{N,B} = \{-1,\ 2\} \cup \left\{ p \mid p \in \mathbb{P},\ \left(\frac{N}{p}\right) = 1,\ p \leq B \right\}$$

In der Praxis gibt man sich nicht explizit eine Schranke B vor, sondern wählt die Mächtigkeit der Faktorbasis.
Für die Größenordnung der Faktorbasis gilt ([Pom, S.31]):

$$|FB_N| = L_N[\,0.5,\ 0.5\,]$$

Die laufzeitoptimale Größe der Faktorbasis hängt jedoch sehr stark von der Implementierung und der Art der benutzten Maschine ab. Deshalb bestimmt man sich durch praktische Tests die "optimale" Mächtigkeit der Faktorbasis für eine Zahl N_1. Dieser Wert dient dann als Ausgangsbasis zur Berechnung "optimaler" Faktorbasisgrößen für andere zu faktorisierende Zahlen. Man berechnet sich mit Hilfe dieser praktisch ermittelten Faktorbasisgröße den Wert $\tilde{c}$, den die $o(1)$-Konstante aus der Formel zur Berechnung von $L_{N_1}[\,0.5,\ 0.5]$ in diesem Fall hat.

$$|opt.FB_{N_1}| = \exp\left((0.5+\tilde{c})\cdot(\log N_1)^{\frac{1}{2}}\cdot(\log\log N_1)^{\frac{1}{2}}\right)$$

$$\iff \quad \tilde{c} = \frac{\ln(|opt.FB_{N_1}|)}{\sqrt{\log(N_1)\cdot\log\log(N_1)}} - 0.5$$

Diesen errechneten Wert $\tilde{c}$ setzt man dann für die $o(1)$-Konstante in die L_N-Funktion für andere zu faktorisierende Zahlen ein. Dadurch erhält man gute Näherungen für die Größe der Faktorbasen. Diese “optimalen” Werte sollten jedoch auch immer wieder praktisch auf Optimalität überprüft werden. Unterscheidet sich die Anzahl der Dezimalstellen von N_1 und der neuen zu fakorisierenden Zahl um mehr als 15 Stellen, so sollte dieser berechnete Wert generell praktisch auf Laufzeitoptimalität überprüft werden. Die Tabelle 3.2 enthält berechnete optimale Faktorbasisgrößen für zu faktorisierende Zahlen von 73 bis 109 Dezimalstellen. Zur Berechnung wurden 3 verschiedene Referenzzahlen und deren optimale Faktorbasisgrößen benutzt. Der Algorithmus sieht folgendermaßen aus:

Algorithmus 3.1 (Bestimmung der Faktorbasis)

Eingabe: $N \in \mathbb{Z}$ (zu faktorisierende Zahl) .

Ausgabe: Faktorbasis , Wurzel von N modulo der Faktorbasiselemente

$i \leftarrow 0$;			(1)
$FB[i++] \leftarrow -1$;			(2)
Bestimme $F \in \mathbb{N}$ (die opt. Größe der FB) ;			(3)
WHILE $(i < F)$			(4)
	Bestimme nächste Primzahl p ;		(5)
	IF $\left(\frac{p}{N}\right) = 1$		(6)
	THEN	$FB[i++] \leftarrow p$;	(7)
Bestimme $\forall p_i \in FB$: $SQRTN[i] = \sqrt{N} \bmod p_i$;			(8)

Schritt (8) stellt eine Vorberechnung dar. Es wird mit dem RESSOL-Algorithmus von Daniel Shanks die Quadratwurzel von N modulo aller Faktorbasiselemente bestimmt und im Array $SQRTN$ abgespeichert ([BucMül, S.25]). Bevor eine neue Faktorbasis erstellt wird, teste ich ab, ob eine Datei mit der richtigen Faktorbasis auf Platte existiert. Stimmt lediglich die Größe der Faktorbasis nicht, so wird diese angepaßt. Die aktuelle Faktorbasis wird dann gegebenenfalls auf Platte gesichert.

Dezimalstellen	$N \approx 10^{90}$	$N \approx 10^{100}$	$N \approx 10^{106}$
73	8654	7502	6818
74	9341	8087	7344
75	10078	8715	7907
76	10868	9387	8510
77	11715	10106	9156
78	12623	10877	9846
79	13595	11701	10584
80	14637	12583	11373
81	15752	13526	12215
82	16946	14534	13116
83	18223	15612	14077
84	19589	16763	15104
85	21050	17993	16200
86	22611	19306	17369
87	24280	20707	18616
88	26063	22203	19945
89	27967	23798	21363
90	30000	25500	22873
91	32169	27314	24483
92	34485	29248	26197
93	36955	31309	28022
94	39590	33505	29966
95	42399	35843	32034
96	45393	38333	34235
97	48584	40984	36576
98	51983	43805	39066
99	55604	46807	41713
100	59460	50000	44527
101	63565	53395	47518
102	67934	57005	50695
103	72583	60843	54070
104	77529	64921	57654
105	82789	69254	61460
106	88383	73857	65500
107	94329	78745	69787
108	100649	83936	74335
109	107365	89445	79161

Tabelle 3.2: Berechnung optimaler Faktorbasisgrößen mit 3 Referenzzahlen

3.4 Wahl der Polynome

Ich beschreibe nun eine Familie von Polynomen zweiten Grades, die an Stelle des ursprünglichen Polynoms benutzt werden können. Der allgemeine Ansatz eines solchen Polynoms sieht folgendermaßen aus.

$$Q(z) = A \cdot z^2 + B \cdot z + C$$

Um die Bedingungen zu ermitteln, unter denen die Funktionswerte des Polynoms Quadrate modulo N sind, formt man das Polynom um. Es gilt :

$$\begin{aligned} Q(z) &= A \cdot z^2 + B \cdot z + \frac{B^2}{4 \cdot A} - \frac{B^2}{4 \cdot A} + C \\ &= A \cdot \left(z^2 + \frac{B}{A} \cdot z + \frac{B^2}{4 \cdot A^2} \right) - \frac{B^2 - 4 \cdot A \cdot C}{4 \cdot A} \\ &= A \cdot \left(z + \frac{B}{2 \cdot A} \right)^2 - \frac{B^2 - 4 \cdot A \cdot C}{4 \cdot A} \end{aligned}$$

Aus

$$B^2 - 4 \cdot A \cdot C = N \quad \text{und} \quad A \equiv D^2 \bmod N \quad \text{für ein } D \in \mathbb{Z} \tag{3.1}$$

folgt

$$Q(z) \equiv H^2 \bmod N \ .$$

Unser Ziel ist es, die Funktionswerte der Polynome über einem gegebenen Siebintervall $[-M, M]$ betragsmäßig möglichst klein zu halten. Auf die Wahl des Intervalls gehe ich bei der Beschreibung der Siebphase (Kapitel 3.5) näher ein. Es gibt mehrere numerische Ansätze, das obengenannte Ziel zu erreichen :

- minimiere $sup|Q(z)| \quad über \quad [-M, M]$
- minimiere $\int\limits_{-M}^{M} |Q(z)| dz$
- minimiere $\int\limits_{-M}^{M} Q^2(z) dz$

Diese führen jedoch (unter Beachtung der Nebenbedingungen (3.1)) im wesentlichen zu den selben reellen Lösungen :

$$\begin{aligned} A_{opt} &= W_1\sqrt{N}M \\ B_{opt} &= 0 \\ C_{opt} &= W_2 M\sqrt{N} \end{aligned}$$

mit $W_1W_2 = -\frac{1}{4}$, $W_1 \in [0.70, 0.75]$ und $W_2 \in [-0.35, 0.30]$. [Sil, S.331]

In der Praxis wählt man die Koeffizienten der benutzten Polynome in der Nähe dieser berechneten optimalen Werte. Dazu benutzt man einen Algorithmus von Robert D. Silverman ([Sil, S.332]). Um die Nebenbedingungen (3.1) zu erfüllen, wählt man A in der Nähe von A_{opt} als Quadrat einer Zahl D. D sollte nicht von einem Faktorbasiselement geteilt werden, da für einen Teiler p_t von D nur eine Lösung für

$$Q(z) \equiv 0 \mod p_t$$

existiert (siehe (3.4)). Dies hätte Nachteile für die nächste Phase (Siebphase). D wird dann akzeptiert, wenn ein Primzahltest (z.B. nach Fermat ([Rie, S.93])) D nicht als zusammengesetzt erkennt. Aus A bestimmt man B so, daß

$$B^2 \equiv N \mod 4 \cdot A \ .$$

Durch diese Wahl kann Bedingung (3.1) erfüllt werden und C kann aus

$$C = \frac{N - B^2}{4 \cdot A}$$

bestimmt werden. C wird jedoch nur zum Testen der Korrektheit der Berechnungen benötigt, da für die Funktionswerte gilt

$$\begin{aligned} Q(z) &= A \cdot \left(z + \frac{B}{2 \cdot A}\right)^2 - \frac{B^2 - 4 \cdot A \cdot C}{4 \cdot A} \\ &\equiv A \cdot \left(z + \frac{B}{2 \cdot A}\right)^2 \mod N \\ &\equiv D^2 \cdot \left(\frac{2 \cdot A \cdot z + B}{2 \cdot D^2}\right)^2 \mod N \\ &\equiv \left(\frac{2 \cdot A \cdot z + B}{2 \cdot D}\right)^2 \mod N \end{aligned}$$

Das Polynom wird also eindeutig durch D bestimmt. Am Anfang berechnet man sich eine Pseudoprimzahl $StartD$ als Startwert für D mit :

$$StartD \approx \sqrt{\frac{\sqrt{N/2}}{M}}, \quad StartD \equiv 3 \bmod 4 \quad \text{und } N^{\frac{StartD-1}{2}} \equiv 1 \bmod StartD \quad (3.2)$$

Dann kann man die Koeffizienten des ersten Polynoms berechnen. Um weitere Polynome zu erzeugen, wird solange 4 auf D addiert, bis D wieder alle Bedingungen aus (3.2) erfüllt sind. Erkennt ein Primzahltest D nicht als zusammengesetzt, berechnet man mit diesem Wert die Koeffizienten des neuen Polynoms. Auf diese Weise kann man viele verschiedene Polynome erzeugen. Die Bestimmung der Koeffizienten sieht dann folgendermaßen aus :

Algorithmus 3.2 (Bestimmung der Koeffizienten)

Eingabe: $N \in \mathbb{Z}$ (zu faktorisierende Zahl), D wie oben beschrieben

Ausgabe: A, B Koeffizienten; Lösungen von $Q(z) \equiv 0 \bmod p_i, p_i \in FB$

	$D2inv \leftarrow (2D)^{-1} \bmod N$ /* zur Berechnung der Funktionswerte */	(1)
	$h_0 \leftarrow N^{(D-3)/4} \bmod D$	(2)
	$h_1 \leftarrow N^{(D+1)/4} \bmod D$	(3)
	$h_2 \leftarrow (2h_1)^{-1}(N - h_1^2)/D \bmod D$	(4)
	$A \leftarrow D^2$	(5)
	$B \leftarrow h_1 + h_2 \cdot D \bmod A$	(6)
IF	$B \equiv 0 \bmod 2$	(7)
THEN	$B \leftarrow A - B$	(8)
FOR $i = 1\ TO\ \#FB$		(9)
	Bestimme Lösungen $NULL1[p_i], NULL2[p_i]$ von $Q(z) \equiv 0 \bmod p_i$	(10)

Zur Berechnung von $(2h_1)^{-1} \bmod D$ nutzt man aus, daß $h_0 \equiv h_1^{-1} \bmod D$ ist. Wegen (3.2) gilt :

$$h_0 \cdot h_1 \equiv N^{\frac{D-3}{4}} \cdot N^{\frac{D+1}{4}} \bmod D \equiv N^{\frac{D-1}{2}} \bmod D \equiv 1 \bmod D$$

Man ersetzt die Inversenberechnung durch eine Addition, ein Shiften nach rechts und eine Multiplikation. Also :

$$(2h_1)^{-1} = h_1^{-1} \cdot 2^{-1} \equiv h_0 \ \cdot \ ((D+1)/2) \ \bmod D$$

Durch Schritt (6) des Algorithmus wird

$$B \equiv h_1 + h_2 \cdot D \bmod D^2$$

gesetzt. Damit gilt :

$$B^2 \equiv h_1^2 + 2 \cdot h_1 \cdot h_2 \cdot D + h_2^2 \cdot D^2 \bmod D^2 \tag{3.3}$$

Da

$$2 \cdot h_1 \cdot h_2 \equiv 2 \cdot h_1 \cdot (2h_1)^{-1}(\frac{N - h_1^2}{D}) \bmod D \qquad \text{(Einsetzen für } h_2\text{)}$$

ist, folgt

$$2 \cdot h_1 \cdot h_2 \cdot D \equiv N - h_1^2 \bmod D^2 \ .$$

Setzt man dieses Ergebnis in Gleichung (3.3) ein so folgt :

$$B^2 \equiv N \bmod D^2, \qquad \text{bzw.} \qquad B^2 \equiv N \bmod A$$

Der Algorithmus führt in Schritt (9) und (10) eine Vorberechnung für die nächste Phase durch. In der Siebphase benötigt man die beiden Lösungen

$$NULL1[p_i],\ NULL2[p_i] \in [0 \ ... \ p_i - 1] \qquad \text{von} \qquad Q(z) \equiv 0 \bmod p_i.$$

In die Formel zur Berechnung dieser Werte

$$NULL1[p_i], NULL2[p_i] \equiv (-B \pm \sqrt{N})(2A)^{-1} \bmod p_i \tag{3.4}$$

gehen die Werte von

$$SQRTN[p_i] = \sqrt{N} \bmod p_i$$

ein. Diese wurden von Algorithmus 3.1 in Schritt (8) berechnet.

Die folgende Abbildung zeigt die Startpolynome, deren Koeffizienten durch den Algorithmus, für verschiedene Siebgrenzen M, berechnet wurden. Das Startpolynom ist das durch *StartD* eindeutig bestimmte Polynom. Die zu faktorisierende Zahl N ist in diesem Beispiel eine 104-stellige Zahl (Kofaktor von p(12497)). (Zur Faktorisierung der Zahl wurde $M = 8 \cdot 10^6$ gewählt.)

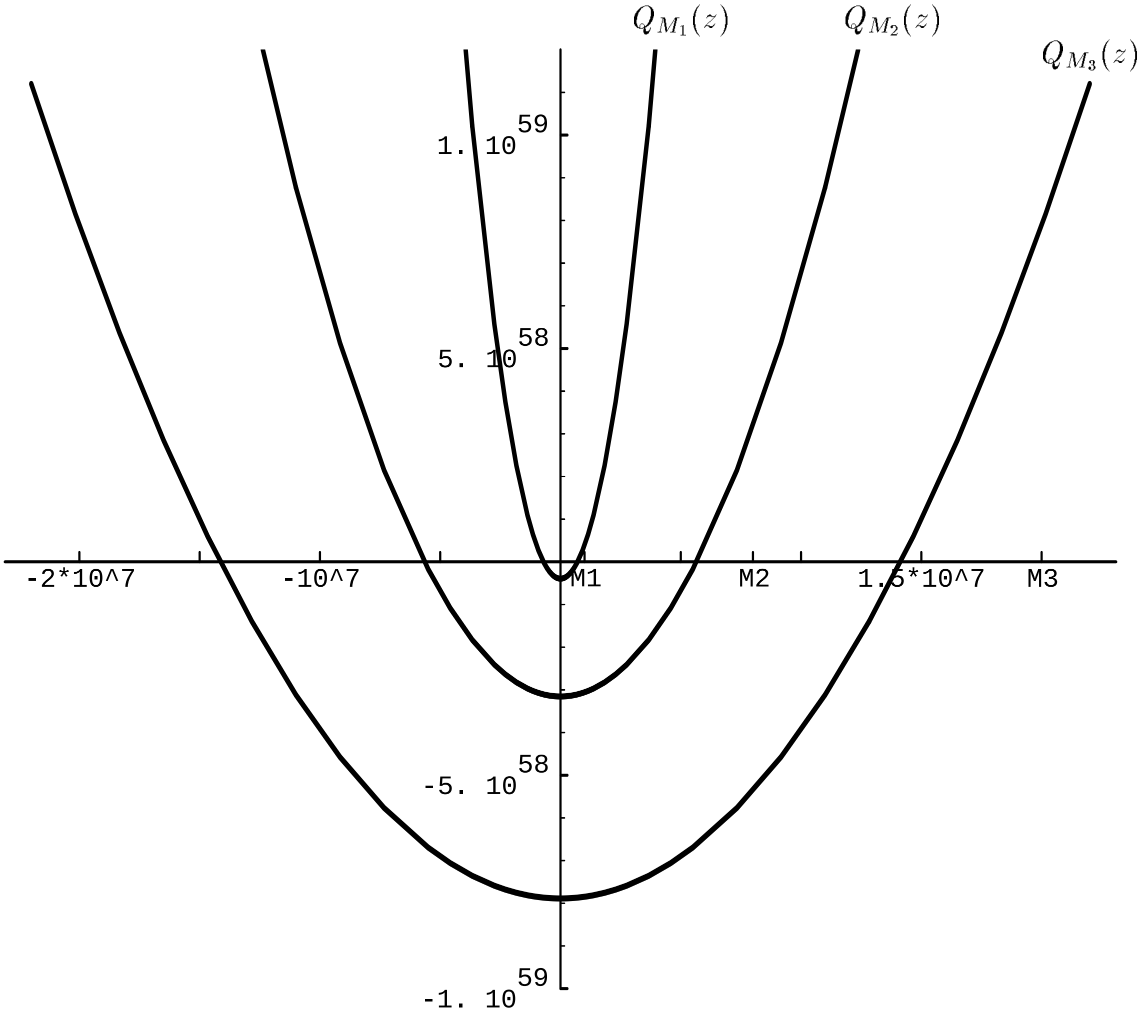


Abbildung 3.1: Berechnete Polynome für verschiedene Siebintervallängen

Der Algorithmus paßt die Polynome an die jeweilige Siebintervallänge (unter Beachtung der Nebenbedingungen (3.1)) an. Dies führt dazu, daß bei größeren Siebintervallen auch die Größe der zu zerlegenden Funktionswerte zunimmt. Dieser Effekt wird besonders deutlich, wenn man die berechneten Startpolynome für $M_1 = 10^6, M_2 = 8 \cdot 10^6$ und $M_3 = 2 \cdot 10^7$ in einem Bild zusammendarstellt.

Durch die Laufzeitkosten, die mit einem Wechsel der Polynome verbunden sind, führen zu kleine Siebintervalle, also zu häufiges Wechseln der Polynome, zu Laufzeitverlusten. Größer werdende Funktionswerte erschweren jedoch deren Zerlegung. Man muß feststellen, wann der größere Aufwand beim Zerlegen der wachsenden Funktionswerte die Laufzeitersparnis durch selteneres Wechseln der Polynome aufbraucht. Ich gehe später noch näher auf dieses Problem ein. (vgl. Kapitel 4.2.3)

3.5 Die Siebphase

In der Siebphase versucht man, Funktionswerte des Polynoms $Q(z)$ mit $z \in [-M, M]$ über der Faktorbasis komplett zu zerlegen. Dazu bestimmt man den Teil der Primfaktorzerlegung der Funktionswerte, der aus Elementen der Faktorbasis gebildet wird.

Eine naive Methode bestünde darin, alle Teiler eines Funktionswertes aus der Faktorbasis mit Probedivision zu bestimmen. Dies ist jedoch sehr laufzeitaufwendig. Man würde

$$\#FB \cdot 2 \cdot M \quad \text{Divisionen einer "langen Zahl" durch eine "kurze Zahl"}$$

benötigen, um für $2 \cdot M$ Funktionswerte festzustellen, welche Elemente der Faktorbasis sie teilen. Außerdem müßte man für jeden Teiler eines Funktionswertes feststellen, wie oft er in dessen Zerlegung auftritt.
Die naive Methode hat Kosten von

$$O(\#FB \cdot M) \quad \text{Divisionen einer "langen" durch eine "kurze Zahl"} \ . \tag{3.5}$$

Beim Quadratischen Sieb wird eine effiziente Methode verwendet, die dem Algorithmus auch den Namen gab. Man nutzt die folgende Beobachtung aus, um sehr schnell die Stellen im Siebarray zu ermitteln, an denen ein Funktionswert von einem Element der Faktorbasis geteilt wird. Ermittelt man diese Stellen für alle Faktorbasiselemente, so hat man auch alle Teiler $p \in FB$ eines Funktionswertes ermittelt.

Für $p \in \mathbb{P}$, $z \in \mathbb{Z}$ und ein quadratisches Polynom $Q(z)$ gilt:

$$\forall_{k \in \mathbb{Z}} \quad Q(z) \equiv Q(z + kp) \bmod p \ .$$

$$\begin{array}{lrcl} \text{Da} & Q(z + k \cdot p) & = & A \cdot (z + k \cdot p)^2 + B \cdot (z + k \cdot p) + C \\ & & = & A \cdot z^2 + 2 \cdot A \cdot z \cdot k \cdot p + A \cdot k^2 \cdot p^2 + B \cdot z + B \cdot k \cdot p + C \\ & & \equiv & A \cdot z^2 + B \cdot z + C \bmod p \\ & & \equiv & Q(z) \bmod p \end{array}$$

Seien nun $z_{1i}, z_{2i} \in [0 \,..\, p_i - 1]$ die Nullstellen des Polynoms $Q(z) \bmod p_i$.
Berücksichtigt man die obige Beobachtung, so kann man alle Stellen im Siebarray bestimmen, an denen ein Funktionswert des Polynoms von p_i geteilt wird. Diese Stellen haben alle die folgende Form :

$$-M \leq x = z_{1i} + k \cdot p, \ x' = z_{2i} + k \cdot p \leq M \quad (\text{mit } k \in \mathbb{Z}).$$

Durch die Berechnung dieser Stellen spart man an jeder Stelle x im Siebarray die Probedivisionen mit Faktorbasiselementen, die kein Teiler des Funktionswertes $Q(x)$

sind. Betrachtet man die Anzahl der Stellen im gesamten Siebarray, die von einem Faktorbasiselement p_i geteilt werden, so gilt :

$$\#\left\{ x \;\middle|\; -M \leq x \leq M, \quad p_i | Q(x) \right\} \approx \frac{4 \cdot M}{p_i}.$$

Mit Hilfe von Satz 1.6 setzt sich die Gesamtlaufzeit der Siebphase dann aus

$$\sum_{i=1}^{\#FB} \frac{4 \cdot M}{p_i} \approx 4 \cdot M \cdot \frac{1}{2} \int_{2}^{p_{max}} \frac{dy}{y \cdot \ln(y)} \approx 2 \cdot M \cdot \ln(\ln(p_{max})) \quad \text{(siehe [RosSch, } S.65])$$

Pomerance benutzt als Abschätzung (vgl. [Pom, S.33])

$$\sum_{i=1}^{\#FB} \frac{4 \cdot M}{p_i} \approx 2 \cdot M \cdot \ln(\ln(\#FB))$$

Im Vergleich zur naiven Methode (3.5) hat diese geschicktere Methode nur Laufzeit von

$O(\ \ln(\ln(\#FB)) \cdot M\ )$ Divisionen einer "langen Zahl" durch eine "kurze Zahl" .

Es gibt jedoch noch eine bessere Methode. Diese Methode läßt die Anzahl der Operationen unverändert, vereinfacht jedoch die Art der Operationen. Um die vergleichsweise teueren Divisionen zu umgehen, arbeitet man mit Logarithmen und Additionen. Man initialisiert das Siebarray mit 0, und addiert an den berechneten Stellen x im Siebarray, an denen ein Faktorbasiselement p_i einen Funktionswert $Q(x)$ teilt, eine ganzzahlige Näherung des dualen Logarithmus von p_i auf. Nach Durchsieben des gesamten Siebarrays steht in jedem Eintrag $SIEB[x]$:

$$SIEB[x] \approx \sum_{i=1}^{\#FB} \log_2(p_i) \cdot e_i \quad \text{mit } e_i = \begin{cases} 1 & \text{falls } p_i | Q(x) \\ 0 & \text{sonst} \end{cases}$$

Der Wert von $SIEB[x]$ ist ein Maß für die Größe der aufmultiplizierten Teiler des Funktionswertes aus der Faktorbasis. Die Laufzeit der Siebphase besteht dann aus

$$\sum_{i=1}^{\#FB} \frac{4 \cdot M}{p_i} \approx 2 \cdot M \cdot \ln(\ln(\#FB))$$

Lesezugriffe auf den Hauptspeicher, kurzen Additionen und Schreibzugriffe auf den Hauptspeicher sowie der Berechnung der Nullstellen des Polynoms modulo aller Faktorbasiselemente, für die gesiebt wird.

Die reine Siebphase hat mit dieser Methode nur Laufzeit von

$$O(\ \ln(\ln(\#FB)) \cdot M\ ) \qquad \text{Additionen kurzer Zahlen}$$

Der Algorithmus hat folgende Form :

Algorithmus 3.3 (Siebphase)

Eingabe: M Siebgrenze, Lösungen z_{1i}, z_{2i} von $Q(z) \equiv 0 \bmod p_i, \quad p_i \in FB$

Ausgabe: Siebintervall nach dem Sieben

FOR $i = -M\ TO\ M$			(1)
	$SIEB[x] \leftarrow 0$ /* Initialisieren des Arrays */		(2)
FOR $p_i \in FB$			(3)
	FORALL $x = z_{1i} + kp_i, \lvert x \rvert \leq M$ /* Sieben mit der 1. Nullstelle */		(4)
		$SIEB[x] \leftarrow SIEB[x] + \log_2(p_i)$	(5)
	FORALL $x = z_{2i} + kp_i, \lvert x \rvert \leq M$ /* Sieben mit der 2. Nullstelle */		(6)
		$SIEB[x] \leftarrow SIEB[x] + \log_2(p_i)$	(7)

Dabei läßt man Potenzen von Faktorbasiselementen unberücksichtigt, obwohl prinzipiell auch mit Potenzen von Faktorbasiselementen gesiebt werden könnte. Aus Laufzeitgründen siebt man jedoch weder mit Potenzen von Faktorbasiselementen noch mit kleinen Faktorbasiselementen (z.B. $p_i < 100$). Dadurch verliert der Siebeintrag etwas an Genauigkeit, man spart jedoch viel Rechenzeit, da der Wert der obigen Summe hauptsächlich von den kleinen Faktorbasiselementen bestimmt wird. Für kleine Faktorbasiselemente p_i würde viel Rechenzeit verbraucht, um einen kleinen Teil ($\log_2(p_i)$) des gesamten Siebeintrags zu bestimmen.

Man ermittelt das Faktorbasiselement p_{start}, ab dem gesiebt wird, durch Untersuchungen, ab wann die Ungenauigkeiten zu Laufzeitproblemen in der Auswertephase (vgl. Kapitel 3.6) führen. Die folgende Tabelle enthält die Laufzeiten der reinen Siebphase in Abhängigkeit vom kleinsten Faktorbasiselement (p_{start}), mit dem gesiebt wird. Das Beispiel wurde für eine 104-stellige Zahl (Kofaktor von p(12497)) mit einer Faktorbasis von 65000 Elementen berechnet.

Die Tabelle enthält auch eine Spalte für

$$sum(p_{start}) := \sum_{i=start}^{65000} \frac{1}{p_i} \quad .$$

$start$	$sum(p_{start})$	$\frac{sum(p_1)}{sum(p_{start})}$ [in %]	Laufzeit [in s]	[in %] von Laufzeit(p_1)
1	1.4909244	100.00	87.00	100.00
5	0.7845582	52.62	45.35	52.12
10	0.6431443	43.14	37.78	43.42
15	0.5693885	38.19	33.18	38.13
20	0.5269719	35.35	30.81	35.41
25	0.4974635	33.37	29.32	33.70
30	0.4729573	31.72	28.18	32.39
35	0.4525252	30.35	27.55	31.66
40	0.4354958	29.21	26.65	30.63
50	0.4112736	27.59	24.36	28.00

Tabelle 3.3: Laufzeit der Siebphase in Abhängigkeit von p_{start}

Das Verhältnis der $sum(p_{start})$ zu $sum(p_1)$ ist in Prozent in der Abbildung 3.2 in Abhängigkeit von p_{start} graphisch dargestellt.

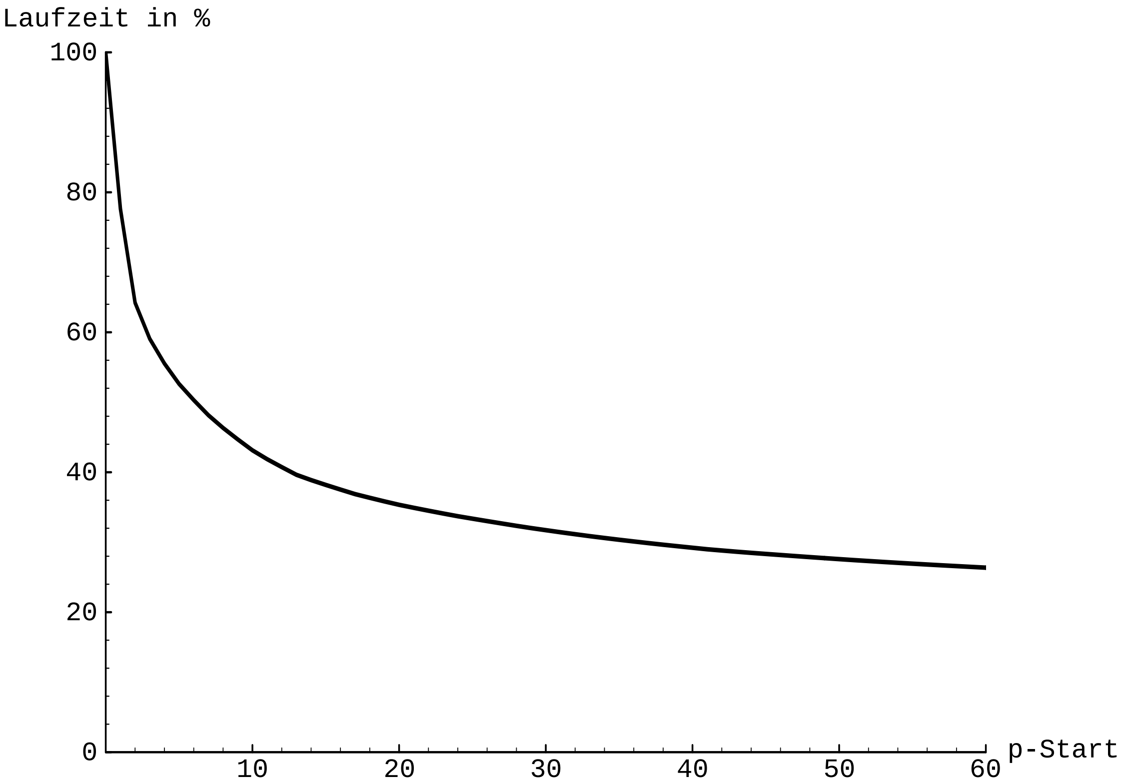


Abbildung 3.2: Laufzeit der Siebphase in Abhängigkeit von p_{start}

Als Anhaltspunkt für die Wahl der Siebgrenze kann folgende Faustformel dienen :

$$M \approx 4 \cdot FB[\#FB]\,.$$

siebintern	Platzbedarf [in MB]	Laufzeit [in s]	rel. Laufzeit
$16 \cdot 10^6$	16.96	24.80	1.00
$8 \cdot 10^6$	9.17	24.82	1.00
$4 \cdot 10^6$	5.31	25.00	1.01
$2 \cdot 10^6$	3.40	25.19	1.02
$1 \cdot 10^6$	2.44	25.70	1.04
$8 \cdot 10^5$	2.20	25.95	1.05
$5 \cdot 10^5$	1.96	26.65	1.07
$2.5 \cdot 10^5$	1.45	28.96	1.16
$1 \cdot 10^5$	1,25	36.37	1.46
$5 \cdot 10^4$	1.20	52.03	2.09
$5 \cdot 10^3$	1.15	402.35	16.22

Tabelle 3.4: Laufzeit der Siebphase in Abhängigkeit von *siebintern*

Die laufzeitoptimalen Parameter hängen jedoch, wie früher schon einmal erwähnt, sehr stark von der Implementierung und den Eigenschaften der verwendeten Maschine ab. In diesem Modul spielt die Geschwindigkeit und die Größe des Hauptspeicherzugriffs eine große Rolle. Ist es möglich, das gesamte Siebarray im Hauptspeicher zu halten bzw. den schnellen Cache-Speicher effizient zu nutzen, so kann ein enormer Speedup des gesamten Programms erzielt werden.

Den Einfluß der Größe des gleichzeitig im Hauptspeicher gehaltenen Siebarrays (*siebintern*) auf die Laufzeit stellt die Tabelle 3.4 dar. Sie enthält Laufzeiten der Siebphase für eine 104-stellige Zahl (Kofaktor von p(12497)). Die gesamte Sieblänge beträgt $16 \cdot 10^6$ ($M = 8 \cdot 10^6$) und *start* wurde gleich 40 gewählt. Sie gibt gleichzeitig den Hauptspeicherbedarf des Programms an, auf den besonders beim Rechnen in inhomogenen verteilten Netzen geachtet werden muß, um Behinderungen anderer Benutzer auszuschließen. In der letzten Spalte sind die Laufzeiten im Verhältnis zur Laufzeit bei $siebintern = 16 \cdot 10^6$ angegeben.

Zu den möglichen Vorberechnungen, die am Anfang des Algorithmus durchgeführt werden, gehört auch das Anlegen des Arrays *LOGP* mit

$$LOGP[i] = (Siebtyp)\lfloor LOGMUL \cdot \log_2(p_i) \rfloor \quad \forall p_i \in FB\ .$$

Um eine möglichst genaue Näherung zu erhalten, multipliziert man die exakten Werte der Logarithmen mit einer Konstanten *LOGMUL*. Diese sollte möglichst groß, gewählt sein, jedoch so, daß es zu keinem Überlauf der aufsummierten Werte in den Siebeinträgen kommt. Durch diese Vorberechnung wird die dauernde Berechnung der Logarithmen beim Sieben gespart. Dies führt vor allem bei vielen Teilintervallen, für die jeweils für alle Faktorbasiselemente diese Näherung berechnet werden müßte, zu starken Laufzeitverbesserungen (Faktor 2-3). Der Hauptspeicherbedarf des Programms erhöht sich jedoch um den Platzbedarf des Array. Da in der Praxis Siebgrenzen M von $7 \cdot 10^6$ (für Zahlen in der Größenordnung 10^{100}) üblich sind, muß das Siebintervall in kleinen Teilen abgearbeitet werden. Im Laufe der Entwickelung des Algorithmus wurde der Datentyp des Siebarrays (*Siebtyp*) von *integer* (4 Byte) auf *unsigned char* (1 Byte) reduziert. Dadurch wurde eine Laufzeitverbesserung von 10% bis 20% (in Abhängigkeit von der Größe der Zahl) erreicht. Dies erklärt sich durch die kleinere Anzahl von Teilintervallen (weniger Overhead beim Sieben) und dem Verkürzen der Länge des Operationstyps.

3.6 Auswerten der Siebphase

In dieser Phase durchsucht man das Siebarray nach Einträgen, die einen berechneten Testwert überschreiten. An diesen Stellen ist der Anteil der Faktorbasiselemente an der Zerlegung des Funktionswertes so hoch, daß eine genauere Untersuchung dieser Funktionswerte sinnvoll ist. Der berechnete Testwert setzt sich aus der bekannten Größenordnung der Polynome (vgl. (2.3)) und einem Korrekturfaktor für die Ungenauigkeit der Siebphase (Sieben ohne Potenzen von Faktorbasiselementen und ohne kleine Faktorbasiselemente) zusammen. Der Testwert wird in Abhängigkeit eines "Toleranzwertes" T definiert als:

$$Testwert(T) := \log\left(\sqrt{N/2} \cdot M\right) - T \cdot \log\left(FB[\#FB]\right), \quad T \in \mathbb{Q} \tag{3.6}$$

Mit dem Wert T kann man die Höhe des Testwertes steuern und somit auch die Anzahl der zu untersuchenden Stellen des Siebarrays. Da die Wahl von T durch Erweiterungen des Grundalgorithmus beeinflußt wird, folgt eine genauere Beschreibung dieses Parameters dort. Die Aufgabe der genaueren Untersuchung der Stellen x im Siebarray mit

$$-M \leq x \leq M : \quad SIEB[x] > Testwert$$

ist es, die Teiler des Funktionswertes $Q(x)$ aus der Faktorbasis und deren Exponenten zu finden. Dies könnte für die relativ wenigen Stellen (ca. 5 pro Million Siebeinträge) durch Probedivision mit allen Faktorbasiselementen geschehen. Diese Operation würde eine Division einer "langen" Zahl (Funktionswert) durch eine "kurze" Zahl (Faktorbasiselement) darstellen. Schneller und geschickter arbeitet folgender Algorithmus:

Algorithmus 3.4 (Auswerten der Siebphase)

Eingabe: M Siebgrenze, Lösungen z_{1i}, z_{2i} von $Q(z) \equiv 0 \bmod p_i, \quad p_i \in FB$

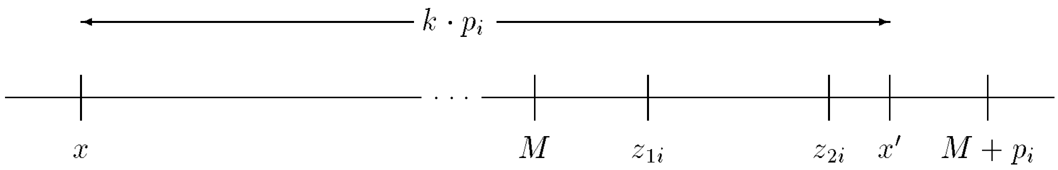
Ausgabe: gefundene Relationen

FOR $x = -M$ TO M	(1)
IF $SIEB[x] > Testwert$	(2)
THEN FOR $i = 1$ TO $\#FB$	(3)
$x_{p_i} \equiv p_i - (M - x) \bmod p_i$	(4)
IF $(z_{1i} == x_{p_i}) \mid\mid (z_{2i} == x_{p_i})$	(5)
THEN Probedivision von $Q(x)$ mit p_i	(6)

Der Algorithmus nutzt die Informationen der Siebphase und ermittelt aus der Differenz der Stelle x und den Nullstellen des Polynoms modulo eines Faktorbasiselememtes p_i, ob der Funktionswert $Q(x)$ von p_i geteilt wird.
Es gilt :

$$p_i | Q(x) \Longleftrightarrow x \equiv z_{1i} \bmod p_i \quad \text{oder} \quad x \equiv z_{2i} \bmod p_i \ .$$

Dieser Test wird folgendermaßen durchgeführt:
Nach Durchsieben des gesamten Siebintervalls enthalten die Werte z_{1i}, z_{2i} die ersten Nullstellen des Polynoms modulo p_i im Teilintervall $[M, M + p_i)$. In Schritt (6) wird die aktuell untersuchte Stelle x in das Intervall $[M, M + p_i)$ "projiziert". Man spart die Berechnung von $z_{1i} \bmod p_i$ und $z_{2i} \bmod p_i$, da man nun die absoluten Werte von z_{1i}, z_{2i} mit dem "projizierten" Wert x' vergleichen kann.



Zerfällt der getestete Funktionswert komplett über der Faktorbasis, so hat man eine Relation gefunden. Diese wird in eine Datei *FAKTORENMATRIX* abgespeichert. Sobald man genügend Relationen gesammelt hat, löst man das lineare Gleichungssystem (2.5) über $\mathbb{F}_2$. Dies geschieht mit Gausselimination. Dazu benutze ich ein "structured Gauss" Programm von Arjen Lenstra, der mir dieses freundlicherweise zur Verfügung gestellt hat. Die Schnittstelle zwischen meinem Programm und dem "structured Gauss" Programm beschreibe ich in der Programmbeschreibung (Kapitel 5) .

3.7 Konstruktion der quadratischen Kongruenzen

Nachdem das lineare Gleichungssystem, das aus den gefundenen Relationen aufgebaut wird (vgl. Kapitel 2.2.2), gelöst ist, muß man die quadratische Kongruenz (vgl. Kapitel 2.1)

$$X^2 \equiv Y^2 \mod N \qquad \text{konstruieren.}$$

Um Teiler der Zahl N zu finden, berechnet man den

$$\mathrm{ggT}(X-Y,N) \quad \text{und den} \ \mathrm{ggT}(X+Y,N) \ .$$

Dabei umgeht man das Wurzelziehen bei der Berechnung von X und Y, indem man X und Y direkt berechnet und nicht zuerst X^2 und Y^2.

Sei l die Anzahl der Relationen, die beim Aufbau des linearen Gleichungssystems benutzt wurden. Das Gleichungssystem hat l Zeilen und eine Lösung ist ein Vektor

$$\lambda = (\ \lambda_1,\ \ldots,\ \lambda_l\ ) \quad \text{mit } \lambda_j \in \{0,1\}, \ \ \text{für} \ \ j \in \{\,1,\ \ldots,\ l\,\}$$

Zur Berechnung von X und Y multipliziert man diejenigen Relationen auf, deren zugehöriges $\lambda_j = 1$ ist. Man berechnet Y aus (vgl. (2.6)):

$$Y^2 \equiv \prod_{p_i \in FB} p_i^{\sum_{j=1}^{l}(e_{i,j}\cdot\lambda_j)} \mod N$$

und damit

$$Y \equiv \prod_{p_i \in FB} p_i^{\frac{\sum_{j=1}^{l}(e_{i,j}\cdot\lambda_j)}{2}} \mod N\ .$$

Man addiert alle Exponenten der Faktorbasiselemente auf und halbiert diese Werte, bevor man die Faktorbasiselemente damit potenziert. Dies ist möglich, da nach Konstruktion die Exponenten jedes Faktorbasiselements gerade sind.

Beim Abspeichern der j-ten Relation der Form (2.4) wird jeweils nur X_j abgespeichert, so daß X aus

$$X \equiv \prod_{j=1}^{l} (X_j)^{\lambda_j} \mod N$$

berechnet werden kann. Nun testet man, ob die Werte des $\mathrm{ggT}(X-Y,N)$ und $\mathrm{ggT}(X+Y,N)$ echte Teiler von N sind. Hat man keine echten Teiler gefunden, so überprüft man, ob die nächste Lösung des Gleichungssystems zu echten Teilern führt. Für jede Lösung des Gleichungssystems beträgt die Wahrscheinlichkeit für das Finden von echten Teilern mindestens 50 %. Hat N mehr als 2 echte Teiler, so erhält man alle Teiler durch Betrachtung weiterer Lösungen.

Kapitel 4

Erweiterungen des Basis-Algorithmus

In diesem Kapitel werde ich Erweiterungen des Basisalgorithmus vorstellen und ihre Auswirkungen auf die Grundphasen und Module erläutern. Obwohl die vorgestellten Erweiterungen die komplexitätstheoretische Laufzeit des Algorithmus nicht verändern, haben sie jedoch zu beachtlichen praktischen Laufzeitverbesserungen geführt. Sie haben die praktischen Einsatzmöglichkeiten dieses Faktorisierungsalgorithmus stark verbessert. Betrachtet man den Bereich, in dem im Moment die Grenzen des praktikablen Einsatzes des Quadratischen Siebes liegen, so stellt man eine Verdoppelung der Laufzeit fest, falls sich die Anzahl der Dezimalstellen der zu faktorisierenden Zahl um 3 Stellen erhöht. Mit der letzten vorgestellten Verbesserung ist es Arjen Lenstra gelungen eine 116-stellige Zahl zu faktorisieren, was den bisherigen Rekord mit dem quadratischen Sieb darstellt.([LenMan][S.15])

4.1 Verwendung eines Multipliers

Das Ziel bei der Verwendung eines *Multipliers* ist es, die Wahrscheinlichkeit, daß ein Funktionswert eines Polynoms innerhalb des Siebarrays über der Faktorbasis zerfällt, zu erhöhen. Diese Wahrscheinlichkeit hängt auch von der Größe der einzelnen Faktorbasiselemente ab, und erhöht sich, wenn viele kleine Primzahlen in der Faktorbasis sind, da diese öfter im Siebarray teilen.
Bei der Bestimmung eines Multipliers k untersucht man die Veränderungen, die auftreten, falls man die zu faktorisierende Zahl N mit der kleinen natürlichen Zahl k multipliziert. Ein zu beobachtender Effekt ist die Vergrößerung der zu faktorisierenden Funktionswerte des Polynoms um den Faktor $\sqrt{k}$. Außerdem ändern sich die Faktorbasiselemente, da für alle Faktorbasiselemente p_i nun gelten muß :

$$\left(\frac{kN}{p_i}\right) = 1, \quad \text{mit } p_i \in FB$$

Man berechnet sich die Güte der Faktorbasis (bzw. ihrer Elemente) für verschiedene Testwerte k, wobei die Vergrößerung der Funktionswerte berücksichtigt wird, und bestimmt so den "optimalen" Wert des Multipliers. Als Testwerte für den Multiplier k werden ungerade Primzahlen und Produkte von Primzahlen, in denen jede Primzahl nur in einfacher Potenz vorkommt, gewählt. Dabei testet man nur Werte, die kleiner einer Schranke sind, und ermittelt die Güte der Faktorbasis für $k = 1$ als Vergleichswert. Für die Menge der Kandidaten gilt :

$$K := \left\{ \prod_{j=1}^{\infty} p_j^{e_j} \leq Schranke \;\middle|\; p_j \in \mathbb{P}, \quad p_j > 2, \quad e_j \in \{0,1\} \right\}.$$

Wählt man als Schranke z.B. 35, so sieht K folgendermaßen aus:

$$K = \{3,\ 5,\ 7,\ 11,\ 13,\ 15,\ 17,\ 19,\ 21,\ 23,\ 29,\ 31,\ 33,\ 35\}.$$

Den Wert der modifizierten Knuth-Schroeppel Funktion [Sil, S.335] benutzt man als Maß für die Güte der Faktorbasis und somit des gewählten Multipliers.

Sei k der geteste Wert des Multipliers und N die zu faktorisierende Zahl. Dann ist die modifizierte Knuth-Schroeppel Funktion definiert als :

$$f(k,N) = \sum_{i=1}^{\#FB} g(p_i, kN) \cdot \log_2(p_i) \;-\; \frac{1}{2}\log_2(k)$$

mit

$$g(p,kN) \;=\; \begin{cases} \frac{2}{p} & \text{falls} \quad p \nmid k \\[2ex] \frac{1}{p} & \text{falls} \quad p \mid k \end{cases}$$

$$g(2,kN) \;=\; \begin{cases} 2 & \text{falls} \quad N \equiv 1 \bmod 8 \\ 0 & \text{sonst} \end{cases}$$

Die Summe in der Knuth-Schroeppel Funktion wird am stärksten durch die kleinen Faktorbasiselemente bestimmt. In der Praxis wertet man sie deshalb nur für die ersten j Faktorbasiselemente aus. In meiner Implementierung habe ich als Schranke für den Multiplier 157 gewählt und untersuche die ersten hundert Elemente der Faktorbasis. Meine praktischen Erfahrungen zeigen, daß größere Multiplier ($k > 11$) sehr selten gewählt werden. Dies gilt besonders für Zahlen mit mehr als 90 Dezimalstellen. Tests mit 70-stelligen Zahlen haben gezeigt, daß der Wert der Knuth-Schroeppel Funktion nicht immer die Wahl des "besten" Multipliers, der zur kürzesten Laufzeit führt, nach sich zieht. Meiner Ansicht nach sollten in der Formel die Größe der zu faktorisierenden Zahl, die Siebgrenzen und Potenzen von Faktorbasiselementen als Teiler berücksichtigt werden. Trotz dieser Kritikpunkte kann man durch geeignete Wahl des Multipliers die Gesamtlaufzeit um den Faktor 2 verkürzen. Es empfiehlt sich, die Wahl eines Multipliers auch an kurzen praktischen Tests mit verschiedenen Kandidaten (mit hohen Funktionswerten) zu überprüfen.

4.2 Large Prime Variante

4.2.1 Idee

Bisher wurden nur komplette Zerlegungen von Funktionswerten über der Faktorbasis gesucht und verwendet. Man bezeichnet die daraus resultierenden Relationen als *full relations* (oder kurz *fulls*). Sie haben folgende Form :

$$X^2 \equiv \prod_{p_i \in FB} p_i^{e_i} \bmod N \ . \tag{4.1}$$

Wie bereits in Kapitel 3.1 erwähnt, ist die Wahrscheinlichkeit, daß ein Funktionswert komplett über der Faktorbasis zerfällt, sehr gering ($\approx 10^{-10}$). Die *Large Prime Variante* verwertet auch "fast" komplette Zerlegungen, die von der Testphase gefunden werden. Zerfällt beim Testen einer Stelle x der Funktionswert $Q(x)$ bis auf einen Rest q (die sogenannte *large prime*) mit

$$p_{max} = max\{p \in FB\} < q < p_{max}^2$$

komplett über der Faktorbasis, so ist q eine Primzahl und man hat eine sogenannte "fast" komplette Zerlegung gefunden. Die Zerlegung sieht folgendermaßen aus :

$$Q(x) = q \cdot \prod_{p_i \in FB} p_i^{e_i} \ . \tag{4.2}$$

Die daraus resultierenden Relationen werden als *partial relations* (oder kurz *partials*) bezeichnet und haben folgende Form :

$$X^2 \equiv q \cdot \prod_{p_i \in FB} p_i^{e_i} \bmod N$$

Findet man nun 2 *partial relations* mit derselben *large prime* q, so kann man diese zu einer *full relation* kombinieren. Aus

$$X_1^2 \equiv q \cdot \prod_{p_i \in FB} p_i^{e_{i,1}} \bmod N$$

und

$$X_2^2 \equiv q \cdot \prod_{p_i \in FB} p_i^{e_{i,2}} \bmod N$$

kann folgende *full relation* erzeugt werden :

$$X^2 \equiv \prod_{p_i \in FB} p_i^{e_i} \bmod N$$

mit

$$X \equiv \left(\frac{X_1}{X_2}\right) \bmod N \quad \text{und} \quad \forall_{i \in \{1,\ldots,\#FB\}} \quad e_i = e_{i,1} - e_{i,2}$$

Die Berechnung von $X_2^{-1} \bmod N$ ist möglich, falls $\mathrm{ggT}(X_2, N) = 1$ ist. (Sonst wurde ein Teiler der zu faktorisierenden Zahl N gefunden.) Im allgemeinen kann man aus $k \geq 2$ paarweise verschiedenen *partial relations* mit der gleichen *large prime* $k-1$ *full relations* erzeugen. Alle anderen möglichen Kombinationen sind linear abhängig von diesen $k-1$ *full relations* und werden deshalb sinnvollerweise nicht in das Gleichungssystem aufgenommen. In der Praxis sammelt man nicht alle *partial relations*, sondern wählt eine obere Schranke $LARGE$ für die *large primes*. Die gesammelten *partial relations* haben dann folgendes Aussehen :

$$X^2 \equiv q \cdot \prod_{p_i \in FB} p_i^{e_i} \bmod N$$

mit

$$p_{max} < q < LARGE < p_{max}^2$$

In meinen bisherigen Faktorisierungen habe ich $LARGE = 10^8$ verwendet. Diese Wahl sorgt dafür, daß alle *large primes* in Integer-Variablen abgespeichert werden können und nicht zu viele *partial relations* mit sehr großen *large primes*, die nur einmal während der kompletten Faktorisierung auftauchen, aufgenommen werden. Diese sind nicht an der Bildung von *full relations* beteiligt und liefern keinen Beitrag zur Faktorisierung. Man findet etwa 10 - 20 mal mehr *partial relations* als *full relations* (in Abhängigkeit von der Größe der Zahl und der Faktorbasis).

Die *Large Prime Variante* führt (um mehr *partial relations* zu finden) zu einer Erhöhung des Toleranzwertes T (vgl. (3.6)) in der Auswertephase. Diese Erhöhung führt auch dazu, daß die Ungenauigkeiten der Siebphase (vgl. Kapitel 3.5) an Bedeutung für das Finden von *full relations* verlieren. Die Laufzeit der Testphase ist proportional zur Anzahl der untersuchten Siebstellen. Somit führt die Erhöhung des Toleranzwertes auch zu einer Erhöhung der Laufzeit der Testphase. Man sucht nun experimentell den Toleranzwert, bei dem die *Large Prime Variante* insgesamt zur größten Laufzeitersparnis führt. Diese kann bei 50 - 60 % liegen.

In der Praxis führt die *Large Prime Variante* zu spürbaren Laufzeitverbesserungen, falls die zu faktorisierenden Zahlen mehr als 60 Dezimalstellen haben. Dabei nimmt ihre Bedeutung mit der Größe der zu faktorisierenden Zahlen zu. Dies spiegelt sich auch in der Wahl des Toleranzwertes, mit dessen Hilfe man die Gewichtung der *Large Prime Variante* steuert. Bei 60-stelligen Zahlen wird der Toleranzwert für das Finden von *full relations* optimiert, und *partial relations* treten quasi nur als Nebenprodukt auf. Bei 70-stelligen Zahlen wird dagegen der Toleranzwert komplett an die *Large Prime Variante* angepaßt, und die *full relations* verlieren an Bedeutung. Die Effektivität der *Large Prime Variante* nimmt mit der Gesamtlaufzeit der Faktorisierung zu. Dies wird durch eine Anlaufphase bedingt. Am Anfang sammelt man hauptsächlich *partial relations* und findet nur wenige *large prime* Paare. Mit der Anzahl der gesammelten Relationen nimmt die Wahrscheinlichkeit zu, Paare zu finden. Der folgende Abschnitt beschäftigt sich mit einer theoretischen Untersuchung der *Large Prime Variante*, an Hand derer dieser Sachverhalt verdeutlicht wird.

4.2.2 Analyse

Um eine Aussage über die Anzahl der zu erwartenden *full relations*, die aus *partial relations* kombiniert werden können, zu erhalten, faßt man das Finden der *partial relations* als Wahrscheinlichkeitsexperiment auf. Zuerst ermittelt man die Menge der Primzahlen P_{LARGE}, die als *large prime* in einer *partial relation* auftreten können.

$$P_{LARGE} := \left\{ q \in \mathbb{P} \;\middle|\; p_{max} = max\{p \in FB\} < q < LARGE < p_{max}^2, \quad \left(\frac{N}{q}\right) = 1 \right\}$$

Da nur etwa die Hälfte aller Primzahlen, die kleiner als $LARGE$ sind, mögliche Teiler eines Funktionswertes eines Polynoms sind (vgl. Kapitel 3.3), gilt für die Mächtigkeit S der Menge P_{LARGE} der möglichen Primzahlen, die in einer *partial relation* vorkommen können :

$$S \approx \frac{1}{2} \cdot \pi(LARGE) - \#FB \quad .$$

Mit Hilfe der Abschätzung von Legendre für die π-Funktion (vgl. Definition 1.7) erhält man :

$$S \approx \frac{1}{2} \left(\frac{LARGE}{\ln(LARGE) - 1.08366} \right) - \#FB \quad .$$

Für eine 95-stellige Zahl (Kofaktor von p(14029)), die mit einer Faktorbasis von 42 000 Elementen und einer Schranke $LARGE = 10^8$ von mir faktorisiert wurde, beträgt $S = 2\,842\,002$. Um die 120-stellige RSA Zahl zu faktorisieren wurde eine Faktorbasis mit 245 810 Elementen und eine Schranke $LARGE = 10^9$ gewählt. Dies führt zu einer Mächtigkeit $S = 25\,212\,949$.

Um ein Gefühl für die Effizienz zu bekommen, betrachtet man zuerst die Anzahl der *partial relations*, die man sammeln muß, um mit einer Wahrscheinlichkeit größer als 1/2 ein Paar zu finden, das zu einer *full relation* kombiniert werden kann. Die Frage, wie oft man aus einer S elementigen Menge mit Zurücklegen ziehen muß, um mit mehr als 50%-iger Wahrscheinlichkeit ein Element zweimal zu ziehen, kann mit Hilfe der Lösung des birthday paradox bestimmt werden(vgl. [Rie, S.177]). Dort stellt man sich die Frage, wieviele Leute man zufällig wählen muß, damit die Wahrscheinlichkeit, das zwei Leute am selben Tag Geburtstag (Tag und Monat) haben, größer als 1/2 ist. Zur Lösung untersucht man das Gegenereignis. Man ermittelt die Wahrscheinlichkeit $Prob_x$ dafür, das x zufällig gewählte Personen an paarweise verschiedenen Tagen Geburtstag haben. Für die erste Person hat man 365 verschiedene Möglichkeiten. Die zweite Person darf nicht am selben Tag wie die erste Person Geburtstag haben, so daß nur noch 364 Möglichkeiten übrig bleiben. Allgemein hat man $365 - (i-1)$ Möglichkeiten für die i-te Person. Es gilt :

$$Prob_x = \frac{365!}{(365-x)! \cdot 365^x} = \prod_{j=1}^{x-1} \frac{365-j}{365} = \prod_{j=1}^{x-1} (1 - \frac{j}{365})$$

Allgemein gilt ([Rie, S.177]) :

$$\prod_{j=2}^{x} (1 - \frac{j-1}{p}) \approx e^{-x(x-1)/2p} .$$

Man bestimmt die Anzahl der benötigten, zufällig gewählten Personen x, in dem man diese Näherung gleich $\frac{1}{2}$ setzt.

$$\begin{array}{rc} & Prob_x \approx e^{-x(x-1)/2p} = \frac{1}{2} \\ \Longrightarrow & \frac{x(x-1)}{2p} = \ln 2 \\ \Longrightarrow & x \approx 1.18 \cdot \sqrt{p} \end{array}$$

Die Lösung des birthday paradox liefert, daß ab 23 zufällig gewählten Personen die Wahrscheinlichkeit, daß zwei von ihnen am gleichen Tag Geburtstag haben größer als 1/2 ist.

Da ich vorerst nur eine Intuition vermitteln will, nehme ich vereinfachend an, daß die Wahrscheinlichkeit, ein Element aus P_{LARGE} zu ziehen (Finden einer *partial relation* mit diesem Element als *large prime*) für alle Elemente gleich groß ist. In den obigen Beispielen müßte man für die 95-stellige Zahl mindestens 1989 *partial relations* sammeln, um mit Wahrscheinlichkeit 1/2 eine *full relation* kombinieren zu können. Für die 120-stellige RSA-Zahl sind dazu mindestens 5926 *partial relations* nötig.

Nachdem man nun berechnen kann, wieviele *partial relations* ungefähr benötigt werden, um mit Wahrscheinlichkeit 1/2 eine *full relation* zu bilden, ist man an dem Erwartungswert der Anzahl der *full relations*, die aus r vielen *partial relations* gebildet werden können, interessiert. Dazu benötigt man eine genauere Abschätzung der Wahrscheinlichkeit, daß eine *large prime* in einer *partial relation* vorkommt. Im allgemeinen kann man davon ausgehen, daß die Wahrscheinlichkeit, daß eine Zahl von einer Primzahl p geteilt wird, umgekehrt proportional zur Größe der Primzahl ist.

Für ein zufällig gewähltes $W \in \mathbb{N}$ gilt :

$$Prob\ [\ p \in \mathbb{P}\ ,\ p|W\ ] \approx \frac{1}{p}$$

Dies gilt, da jede p-te natürliche Zahl von p geteilt wird. Die in diesem Fall betrachteten Zahlen stellen aber eine spezielle Auswahl aller Zahlen dar. Man betrachtet Funktionswerte in der Größenordnung von $O(\sqrt{N})$, die die Eigenschaft haben, bis auf eine *large prime* komplett über der Faktorbasis zu zerfallen. In dieser Teilmenge ist die Wahrscheinlichkeit für größere *large primes* höher, größter Primfaktor einer Zahl zu sein. Ein weiteres algorithmisch bedingtes Problem liegt in der Auswertung der Siebphase. Dort werden nicht alle *partial relations* aus einem Siebintervall gefunden. Um die Anzahl der erfolglos untersuchten Stellen im Siebintervall und damit die Laufzeit pro *partial relation* klein zu halten, sucht man nur diejenigen *partial relations*, die zu Stellen mit höheren Siebwerten korrespondieren. Dies sind in der Regel *partial relations* mit kleineren *large primes*.

Mit Hilfe der μ-Funktion kann man Aussagen über die Wahrscheinlichkeit treffen, daß der größte Primfaktor und der zweitgrößte Primfaktor einer Zahl unterhalb von Schranken liegen (vgl. Definition 1.5). Die μ-Funktion kann benutzt werden, um die Wahrscheinlichkeit, daß eine *partial relation* gefunden wird, zu berechnen. Man kann jedoch keine Aussage über die Wahrscheinlichkeit, daß eine spezielle *large prime* in einer *partial relation* auftaucht treffen. Dies ist ohne ausführliche und genaue zahlentheoretische Untersuchung nicht möglich. Man wählt deshalb die Näherung für zufällig gewählte Zahlen und berücksichtigt die spezielle Art der Zahlen durch einen Gewichtungsfaktor γ (vgl. [LenMan, S.11]). Für alle Primzahlen $q \in P_{LARGE}$ setzt man die Wahrscheinlichkeit $Prob_q$, daß q als *large prime* in einer *partial relation* vorkommt auf

$$Prob_q = \frac{1/q^\gamma}{\sum_{p \in P_{LARGE}} 1/p^\gamma} \quad \text{mit } \frac{1}{2} < \gamma < 1\ .$$

Das Finden von *partial relations* wird jetzt mit dem Ziehen von *large primes* aus P_{LARGE} mit Zurücklegen assoziiert. Um den Erwartungswert für die Anzahl der kombinierten *full relations* angeben zu können, benötigt man die Anzahl der in Vielfachheit in den r *partial relations* vorkommenden *large primes*. Dazu bestimmt man den Erwartungswert $E(X)$ für die Anzahl der Elemente aus P_{LARGE}, die in den r *partial relations* nicht auftreten. Für diesen gilt :

$$E(X) = \sum_{q \in P_{LARGE}} (1 - Prob_q)^r\ .$$

Aus der Gesamtzahl der *large primes* (S) und dem Erwartungswert $E(X)$ kann man einen Erwartungswert $E(T)$ für die Anzahl der in den r *partial relations* vorkommenden *large primes* angeben.

$$E(T) = S - E(X)$$

Diese *large primes* verteilen sich auf die r *partial relations*. Deshalb gibt es

$$E(Anzahl) = r - E(T)$$

partial relations, in denen *large primes* vorkommen, die bereits einmal aufgetreten sind und somit zu *full relations* führen. Dabei wird die *partial relation*, in der die *large prime* zum ersten Mal aufgetreten ist, nicht mitgezählt. Der Erwartungswert $E(Anzahl)$ für die Anzahl der *full relations*, die aus den r *partial relations* gebildet werden können, berechnet sich dann folgendermaßen :

$$E(Anzahl) = r - S + \sum_{q \in P_{LARGE}} (1 - Prob_q)^r \; .$$

Die Anzahl, der aus r *partial relations* konstruierbaren *full relations* liegt in $O(r^2)$ (vgl. [LenMan, S.12]). Mit Hilfe einiger Abschätzungen kann man eine schnell berechenbare Funktion angeben, die in Abhängigkeit von der Anzahl r der *partial relations* und dem Gewichtungsfaktor γ eine gute Näherung für diesen Erwartungswert liefert.

Die Berechnung sieht im einzeln folgendermaßen aus :

- Bei der Berechnung von S wird die Anzahl der Primzahlen, die kleiner als $LARGE$ sind, mit Hilfe numerischer Integration aus der Abschätzung für die π-Funktion von Gauss (vgl. Definition 1.7) bestimmt. Halbiert man diese Anzahl (nur die Hälfte aller Primzahlen sind mögliche Teiler von Funktionswerten) und subtrahiert die Größe der Faktorbasis, so erhält man eine Näherung für S.

$$S \approx \frac{1}{2} \cdot \int_2^{LARGE} \frac{dx}{\ln(x)} \quad - \quad \#FB$$

- Bei der Berechnung von $Prob_q$ benutzt man Satz 1.6 und erhält

$$\sum_{p \in \mathbb{P},\; p<x} 1/p^\gamma \approx \int_2^x \frac{dy}{y^\gamma \cdot \ln(y)}$$

  und damit

$$\sum_{p \in P_{LARGE}} 1/p^\gamma \approx \frac{1}{2} \cdot \int_{p_{max}}^{LARGE} \frac{dy}{y^\gamma \cdot \ln(y)} \quad .$$

Für $Prob_{q,\gamma}$ gilt dann :

$$Prob_{q,\gamma} := \frac{1/q^\gamma}{\sum_{p \in P_{LARGE}} 1/p^\gamma} \approx \frac{2}{q^\gamma \cdot \int\limits_{p_{max}}^{LARGE} \frac{dy}{y^\gamma \cdot \ln(y)}} .$$

- Aus Satz 1.6 und der Tatsache, daß nur die Hälfte aller Primzahlen mögliche Teiler eines Funktionswertes sind, folgt :

$$\sum_{q \in P_{LARGE}} (1 - Prob_{q,\gamma})^r \approx \frac{1}{2} \cdot \int\limits_{p_{max}}^{LARGE} \frac{\left(1 - 2 \cdot \left(z^\gamma \cdot \int\limits_{p_{max}}^{LARGE} \frac{dy}{y^\gamma \cdot \ln(y)}\right)^{-1}\right)^r}{\ln(z)} \, dz .$$

Damit ergibt sich für den Erwartungswert der Anzahl der konstruierbaren *full relations* aus *partial relations* in Abhängigkeit von r und γ :

$$E_{r,\gamma} \approx r + \#FB + \frac{1}{2} \left(\int\limits_{p_{max}}^{LARGE} \frac{\left(1 - 2 \cdot \left(z^\gamma \cdot \int\limits_{p_{max}}^{LARGE} \frac{dy}{y^\gamma \cdot \ln(y)}\right)^{-1}\right)^r}{\ln(z)} \, dz - \int\limits_{2}^{LARGE} \frac{dx}{\ln(x)} \right) .$$

Die so ermittelte Näherung für den Erwartungswert der konstruierbaren *full relations* ist recht genau. Anhand von 3 faktorisierten Zahlen in verschiedenen Größenordnungen wurden die theoretisch ermittelten Werte mit den praktisch erzielten Werten verglichen. Dabei handelt es sich einmal um eine 84-stellige Zahl (Kofaktor von $31^{74} + 1$) aus dem erweiterten Cunningham Projekt ([BreRie]), die ich für Herman J.J. te Riele faktorisiert habe. Außerdem wurden die Ergebnisse mit den Daten, die bei der Faktorisierung einer 95-stelligen Zahl (Kofaktor von p(14029)) und einer 102-stelligen Zahl (Kofaktor von p(15031)) gesammelt wurden, verglichen. In allen Fällen wurde $LARGE = 10^8$ gewählt. Die Mächtigkeit der Faktorbasis betrug 23 750 (84-stellige Zahl), 42 000 (95-stellige Zahl) und 60 000 (102-stellige Zahl). Die Tabellen enthalten Werte für $E(Anzahl)_{r,\gamma}$ für verschiedene γ und r. Für alle drei Zahlen liefert $\gamma = 0.775$ gute Näherungen. In [LenMan, S.11] wird ein größeres Intervall für γ ($\gamma \in [2/3\,,\,3/4]$) angegeben. In Abbildung 4.2 sind die beobachteten Werte gestrichelt eingezeichnet.

4.2.3 Vergleich des Findens von *partial* und *full relations*

Wie bereits früher erwähnt, kann man mit Hilfe der ρ-Funktion die Wahrscheinlichkeit des Findens einer *full relation* angegeben werden (vgl. Kapitel 3.1). Um mit Hilfe der ρ-Funktion Aussagen über die Wahrscheinlichkeit des Findens einer *full relation*

Vergleich für 84-stellige Zahl, $|FB| = 23\,750$

partial relations	Praxis	$\gamma = 0.75$	$\gamma = 0.775$	$\gamma = 0.8$
18612	182	188	202	217
30313	486	476	512	551
45124	1084	1030	1107	1191
60896	1955	1845	1982	2130
74564	2945	2733	3134	2932
84744	3767	3500	3753	4026
100752	5234	4884	5232	5608
109108	6103	5691	6093	6527
115255	6761	6320	6764	7243
124045	7787	7272	7779	8325
128747	8337	7806	8349	8932
129368	8402	7878	8425	9013
129781	8445	7926	8476	9067

Tabelle 4.1: Vergleich des Erwartungswert und der Anzahl kombinierbarer *fulls*

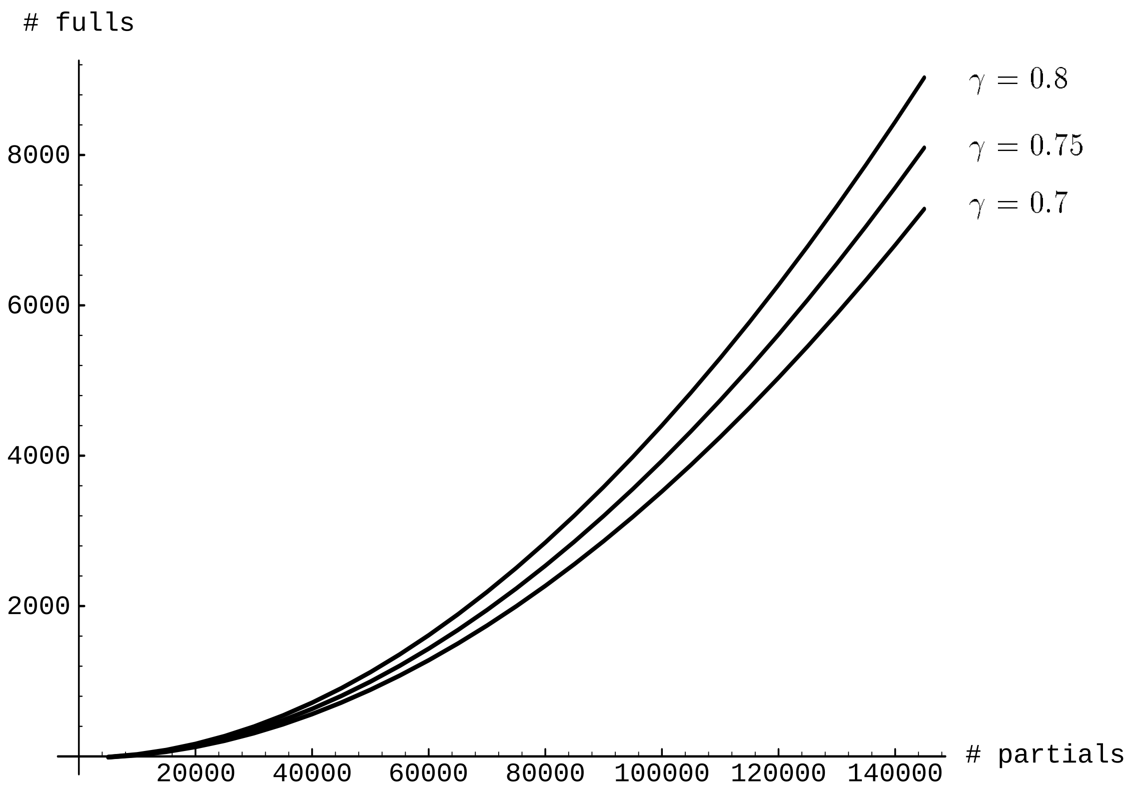


Abbildung 4.1: $E(Anzahl)_{r,\gamma}$ für $\gamma = 0.7, 0.75$ und 0.8 für Kofaktor von p(14029)

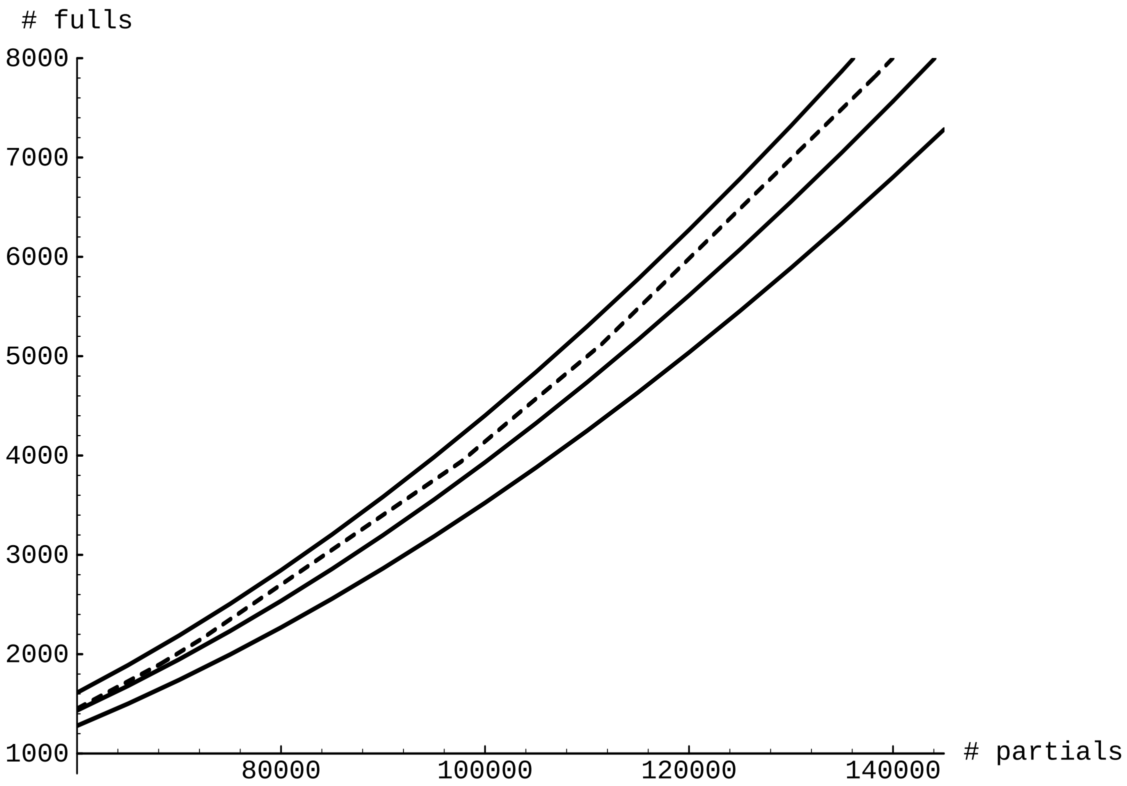


Abbildung 4.2: Ausschnitt von Abbildung 4.1 mit den beobachteten Werten

Vergleich für 95-stellige Zahl, $|FB| = 42\,000$

partial relations	Praxis	$\gamma = 0.7$	$\gamma = 0.75$	$\gamma = 0.775$	$\gamma = 0.8$
5645	18	0	0	0	0
35389	514	437	492	523	556
42024	739	623	700	743	790
57779	1329	1186	1329	1409	1495
68171	1901	1651	1848	1958	2076
72139	2149	1848	2067	2190	2321
97667	3939	3364	3754	3972	4204
103882	4476	3798	4236	4480	4741
111370	5115	4353	4852	5130	5427
138286	7821	6641	7387	7801	8243
141842	8197	6977	7759	8192	8655

Tabelle 4.2: Vergleich des Erwartungswert und der Anzahl kombinierbarer *fulls*

Vergleich für 102-stellige Zahl, $|FB| = 60\,000$

partial relations	Praxis	$\gamma = 0.75$	$\gamma = 0.775$	$\gamma = 0.8$
11536	67	0	0	0
37251	480	444	469	497
54692	1069	1017	1071	1129
56024	1107	1069	1126	1187
82546	2372	2363	2483	2611
86002	2592	2566	2696	2835
88345	2743	2708	2845	2991
95157	3195	3142	3300	3468
99934	3518	3465	3637	3822
102348	3691	3633	3814	4006
104648	3878	3797	3985	4186
107094	4055	3975	4172	4382
109405	4248	4147	4352	4570
111970	4452	4342	4556	4784
116200	4824	4672	4901	5146
118541	5030	4860	5098	5352
120855	5234	5048	5295	5559
123245	5460	5247	5503	5776
125580	5677	5444	5710	5992
127965	5876	5650	5925	6217
130263	6062	5851	6135	6437
132659	6277	6064	6358	6671
134987	6480	6274	6578	6901
137402	6686	6496	6810	7144
142120	7167	6940	7274	7630
144600	7403	7179	7524	7891
146829	7647	7397	7752	8129
151551	8107	7868	8244	8644
153938	8344	8112	8499	8910
156150	8609	8340	8738	9160
158027	8798	8537	8943	9375
160908	9135	8842	9262	9708
165480	9632	9337	9779	10249
167975	9908	9612	10067	10549
170350	10182	9878	10344	10839
172647	10458	10138	10615	11123
175003	10728	10408	10897	11417
177232	10979	10666	11167	11699
179688	11283	10954	11468	12013
180879	11433	11095	11615	12166

Tabelle 4.3: Vergleich des Erwartungswert und der Anzahl kombinierbarer *fulls*

zu erhalten, benötigt man das größte Faktorbasiselement p_{max} und die Größenordnung der Funktionswerte, über die gesiebt wird. Folgende Betrachtungen führen zu dieser Größenordnung für die multipolynomielle Version des quadratischen Siebes. Sei N die zu faktorisierende Zahl, k der gewählte Multiplier und M die Siebgrenze. Dann gilt für die Größenordnung der Funktionswerte :

$$|Q(z)| < \frac{M \cdot \sqrt{kN}}{2 \cdot \sqrt{2}}, \qquad \text{für} \qquad z \in [-M, M].$$

Die Extremwerte des Polynoms $Q(z)$ liegen in diesem Intervall bei $z_1 = 0$ und $z_2 = \pm\, M$.

An diesen Stellen gilt :

$$Q(0) = C \approx -\frac{M \cdot \sqrt{kN}}{2 \cdot \sqrt{2}}$$

und

$$Q(\pm\, M) = A \cdot M^2 + C \approx \frac{\sqrt{2} \cdot \sqrt{kN}}{2 \cdot M} \cdot M^2 - \frac{M \cdot \sqrt{kN}}{2 \cdot \sqrt{2}} = \frac{M \cdot \sqrt{kN}}{2 \cdot \sqrt{2}}$$

Damit sind die Funktionswerte, im Vergleich zur Grundversion des Quadratischen Siebes (vgl. 2.3), um den Faktor $\sqrt{8}$ kleiner.

Nun kann man, wie bereits in Kapitel 3.1 für die Grundversion des quadratischen Siebes gezeigt wurde, die Wahrscheinlichkeit des Findens einer *full relation* angeben. Man setzt (vgl. Definition 1.4)

$$L = \frac{M \cdot \sqrt{kN}}{2 \cdot \sqrt{2}}$$

und bestimmt aus

$$\begin{aligned} L^{\frac{1}{\alpha}} &= p_{max} \\ \Longleftrightarrow \quad \alpha &= \frac{\ln(L)}{\ln(p_{max})} = \frac{\ln(M) + 0.5 \cdot \ln(kN) - \ln(\sqrt{8})}{\ln(p_{max})} \end{aligned}$$

das Argument der ρ-Funktion.

Dieser Wert wird auch bei der Bestimmung der Wahrscheinlichkeit für das Finden einer *partial relation* benötigt. Dazu benutzt man die μ-Funktion (vgl. Definition 1.5). Zusätzlich wird aus der Schranke für die *large prime* ($LARGE$) und dem größten Faktorbasiselement (p_{max}) der Wert für β berechnet. Dieser ist unabhängig von der Größenordnung der zu faktorisierenden Funktionswerte der Polynome.

Stellen	α	β	$\rho(\alpha)$	Praxis	$\mu(\alpha,\beta)$	Praxis
84	8.467	1.403	$6.524 \cdot 10^{-9}$	$1.426 \cdot 10^{-8}$	$1.363 \cdot 10^{-7}$	$1.960 \cdot 10^{-7}$
91	8.677	1.340	$3.158 \cdot 10^{-9}$	$5.068 \cdot 10^{-9}$	$5.349 \cdot 10^{-8}$	$6.289 \cdot 10^{-8}$
95	8.963	1.334	$1.158 \cdot 10^{-9}$	$1.838 \cdot 10^{-9}$	$2.012 \cdot 10^{-8}$	$4.514 \cdot 10^{-8}$
97	9.082	1.303	$7.602 \cdot 10^{-10}$	$1.905 \cdot 10^{-9}$	$1.179 \cdot 10^{-8}$	$2.498 \cdot 10^{-8}$
102	9.376	1.290	$2.664 \cdot 10^{-10}$	$3.295 \cdot 10^{-10}$	$4.091 \cdot 10^{-9}$	$4.667 \cdot 10^{-9}$
104	9.465	1.283	$2.006 \cdot 10^{-10}$	$3.647 \cdot 10^{-10}$	$2.918 \cdot 10^{-9}$	$4.767 \cdot 10^{-9}$
120	9.807	1.322	$5.610 \cdot 10^{-11}$	$6.459 \cdot 10^{-11}$	$1.024 \cdot 10^{-9}$	$2.456 \cdot 10^{-9}$

Tabelle 4.4: Vergleich berechneter μ und ρ Werte mit der Praxis

Aus

$$\begin{aligned} & L^{\frac{\beta}{\alpha}} &=& \; LARGE \\ \iff & \frac{\beta}{\alpha} &=& \; \frac{\ln(LARGE)}{\ln(L)} \\ \iff & \beta &=& \; \frac{\ln(LARGE)\cdot\ln(L)}{\ln(L)\cdot\ln(p_{max})} \;=\; \frac{\ln(LARGE)}{\ln(p_{max})} \end{aligned}$$

erhält man den zweiten Parameter der μ-Funktion.

Vergleicht man die auf diese Weise theoretisch ermittelten Werte mit praktischen Erfahrungen, so treten teilweise starke Unterschiede auf.

Dies kann einmal durch die Verwendung eines Multipliers (vgl. Kapitel 4.1), der zu vielen kleinen Primzahlen in der Faktorbasis und damit zu kleinen möglichen Teilern der Funktionswerte führt, erklärt werden. Eine wichtigere Fehlerquelle liegt in der Tatsache, daß die Funktionswerte mit ihrem Maximum abgeschätzt wurden. Eine genauere Untersuchung der Wahrscheinlichkeit des kompletten Zerfallens von Funktionswerten über der Faktorbasis im Siebintervall zeigt, daß dort sehr große Unterschiede existieren. Für ein Polynom mit optimal gewählten Parametern (vgl. Kapitel 3.4) gilt :

$$Q(z) = \frac{\sqrt{N}}{\sqrt{8}\cdot M}\cdot\left(2\cdot z^2 - M^2\right) \; .$$

Am Beispiel einer 84-stelligen Zahl wird die Verteilung der Wahrscheinlichkeit, eine *full relation* zu finden, genauer untersucht. Als Siebintervallgrenze wurde $3 \cdot 10^6$

gewählt. Aus Symmetriegründen wird nur die rechte Hälfte des Siebintervalls betrachtet. Die Tabelle 4.5 enthält Durchschnittswerte der ρ-Funktion in den einzelnen Teilintervallen. Die Länge der Teilintervalle beträgt 50 000. Zur Berechnung der Durchschnittswerte wurde mit Schrittlänge 25 der Funktionswert des Polynoms und daraus der Wert der ρ-Funktion berechnet.

Intervall	ρ Wert	Intervall	ρ Wert	Intervall	ρ Wert
1	$6.54657 \cdot 10^{-9}$	21	$7.01140 \cdot 10^{-9}$	41	$1.23790 \cdot 10^{-8}$
2	$6.54657 \cdot 10^{-9}$	22	$7.06729 \cdot 10^{-9}$	42	$1.51453 \cdot 10^{-8}$
3	$6.54657 \cdot 10^{-9}$	23	$7.13155 \cdot 10^{-9}$	43	$2.34290 \cdot 10^{-8}$
4	$6.54657 \cdot 10^{-9}$	24	$7.19889 \cdot 10^{-9}$	44	$1.44162 \cdot 10^{-8}$
5	$6.55494 \cdot 10^{-9}$	25	$7.27349 \cdot 10^{-9}$	45	$1.19894 \cdot 10^{-8}$
6	$6.56923 \cdot 10^{-9}$	26	$7.35464 \cdot 10^{-9}$	46	$1.07662 \cdot 10^{-8}$
7	$6.57647 \cdot 10^{-9}$	27	$7.44606 \cdot 10^{-9}$	47	$9.96564 \cdot 10^{-9}$
8	$6.59197 \cdot 10^{-9}$	28	$7.54475 \cdot 10^{-9}$	48	$9.37853 \cdot 10^{-9}$
9	$6.60980 \cdot 10^{-9}$	29	$7.65439 \cdot 10^{-9}$	49	$8.91907 \cdot 10^{-9}$
10	$6.62636 \cdot 10^{-9}$	30	$7.77785 \cdot 10^{-9}$	50	$8.54478 \cdot 10^{-9}$
11	$6.64677 \cdot 10^{-9}$	31	$7.91534 \cdot 10^{-9}$	51	$8.22906 \cdot 10^{-9}$
12	$6.66996 \cdot 10^{-9}$	32	$8.07039 \cdot 10^{-9}$	52	$7.95808 \cdot 10^{-9}$
13	$6.69532 \cdot 10^{-9}$	33	$8.24738 \cdot 10^{-9}$	53	$7.72112 \cdot 10^{-9}$
14	$6.72242 \cdot 10^{-9}$	34	$8.45136 \cdot 10^{-9}$	54	$7.51032 \cdot 10^{-9}$
15	$6.75517 \cdot 10^{-9}$	35	$8.68881 \cdot 10^{-9}$	55	$7.32171 \cdot 10^{-9}$
16	$6.78837 \cdot 10^{-9}$	36	$8.97211 \cdot 10^{-9}$	56	$7.15088 \cdot 10^{-9}$
17	$6.82604 \cdot 10^{-9}$	37	$9.31603 \cdot 10^{-9}$	57	$6.99459 \cdot 10^{-9}$
18	$6.86558 \cdot 10^{-9}$	38	$9.74773 \cdot 10^{-9}$	58	$6.85153 \cdot 10^{-9}$
19	$6.91009 \cdot 10^{-9}$	39	$1.03134 \cdot 10^{-8}$	59	$6.71886 \cdot 10^{-9}$
20	$6.95886 \cdot 10^{-9}$	40	$1.11111 \cdot 10^{-8}$	60	$6.59620 \cdot 10^{-9}$

Tabelle 4.5: Durchschnittswerte der ρ-Funktion in Teilsiebintervallen

Die Werte der Tabelle 4.5 sind in der Abbildung 4.3 graphisch dargestellt. Die Abbildung 4.4 stellt den Betrag des Polynoms $Q(z)$ in der rechten Siebhälfte dar. Man

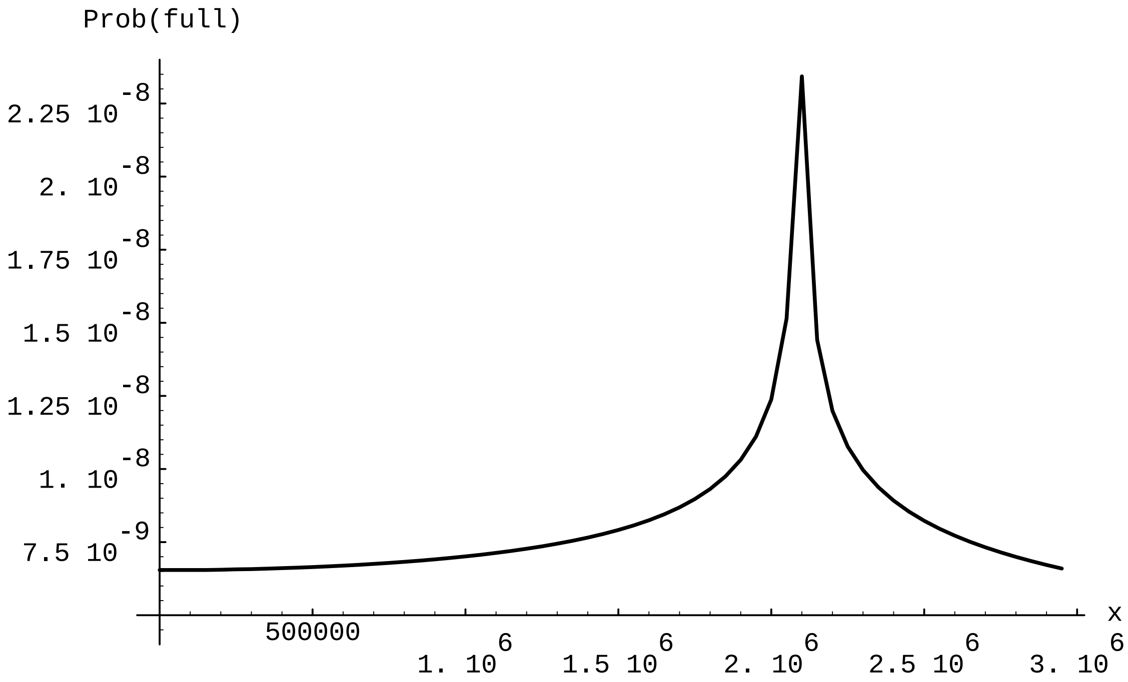


Abbildung 4.3: Verlauf der ρ-Funktion im rechten Teilsiebintervall

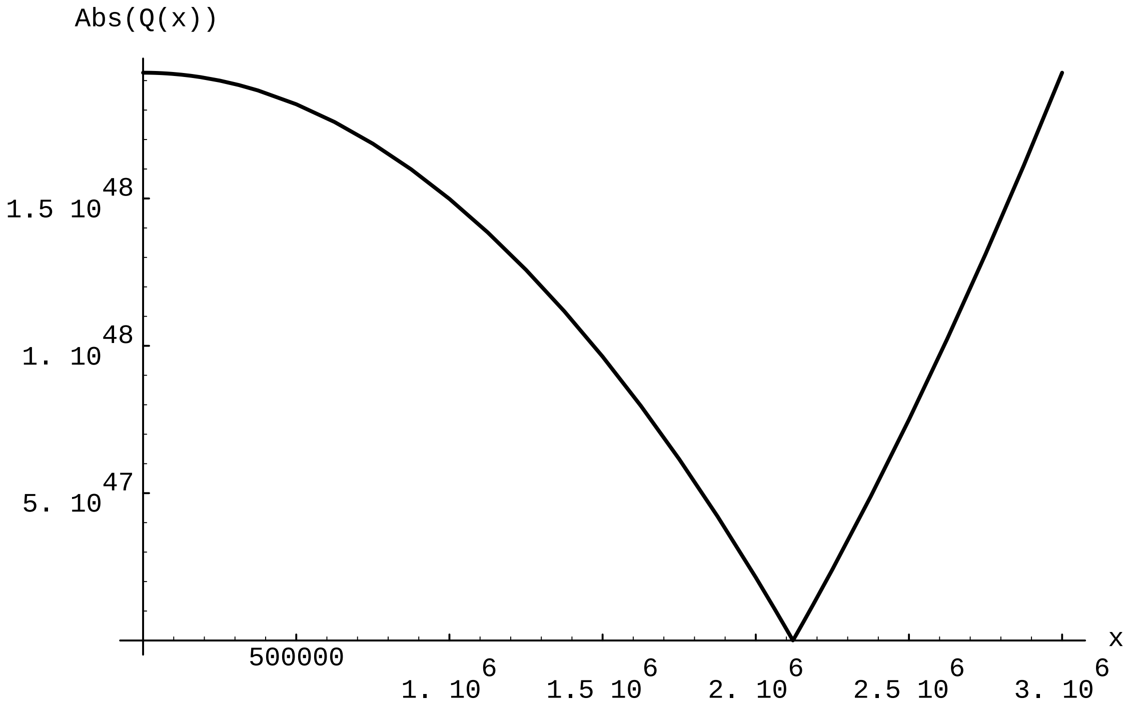


Abbildung 4.4: Betrag des verwendeten Polynoms im rechten Teilsiebintervall

sieht an der Abbildung 4.3 sehr schön, daß die Wahrscheinlichkeit, eine *full relation* zu finden, um die Nullstelle des Polynoms wesentlich höher als im übrigen Siebintervall ist. Da die Abbildung mit Durchschnittswerten aufgestellt wurde, wird der Unterschied noch deutlicher, wenn man die maximalen und minimalen Werte der ρ Funktion im dargestellten Siebintervall betrachtet.

Für die Extremwerte gilt :

$$\rho_{min} = 6.5492 \cdot 10^{-9} \ , \quad \rho_{max} = 3.0023 \cdot 10^{-7} \ .$$

Direkt in der Umgebung der Nullstelle ist die Wahrscheinlichkeit, eine *full relation* zu finden, ungefähr 50 mal höher als am Rand oder in der Mitte des gesamten Siebintervalls. Entfernt man sich von der Nullstelle, so werden die Funktionswerte des Polynoms schnell größer und die Unterschiede in der Wahrscheinlichkeit des Findens einer *full relation* nehmen ab. Es erscheint daher sinnvoll, nur in einem kleinen Intervall um die Nullstellen zu sieben, da dort die Wahrscheinlichkeit des Findens einer *full relation* wesentlich höher als im übrigen Siebintervall ist. Dabei darf man aber weder den höheren Aufwand der Siebphase und den höheren Aufwand durch das häufigere Wechseln des Polynoms noch die Alternative, direkt ein kleineres Siebintervall zu wählen, vergessen. Vorerst vernachlässigt man die algorithmischen Nachteile und untersucht beide Möglichkeiten (Sieben um die Nullstelle und Wahl eines entsprechend kleineren Siebintervalls).

Dazu betrachtet man das Integral über den Beträgen der Funktionswerte. Für ein Polynom Q_{M_1} mit optimal gewählten Koeffizienten in Abhängigkeit einer Siebgrenze M_1 gilt :

$$\int_{-M_1}^{M_1} |Q_{M_1}(z)| \, dz = \frac{2M_1^2\sqrt{N}}{3} - \frac{M_1^2\sqrt{N}}{3\sqrt{2}} = \frac{\sqrt{N} \cdot M_1^2}{3 \cdot \sqrt{2}} \cdot (\sqrt{8} - 1) \ . \tag{4.3}$$

Damit hat man einen Wert für das Sieben in einem kompletten Siebintervall der Länge $2 \cdot M_1$. Nun berechnet man einen Vergleichswert für das Sieben nur in den zwei Teilintervallen um die Nullstellen. Dazu untersucht man zuerst, wie man die Grenzen dieses Siebintervalls um eine Nullstelle wählen muß, damit bei vorgegebener Intervallänge das Integral der Beträge der Funktionswerte minimal wird. Für ein Polynom Q_{M_2} mit optimal gewählten Koeffizienten in Abhängigkeit einer Siebgrenze M_2 gilt :

$$Q_{M_2}(z) = 0 \Longleftrightarrow z = \pm\frac{M_2}{\sqrt{2}} \ .$$

Um das Verhältnis des Teilintervalls links von der Nullstelle und des Teilintervalls rechts von der Nullstelle zu bestimmen, löst man

$$\int_{\frac{-M_2}{\sqrt{2}}-l}^{\frac{-M_2}{\sqrt{2}}+r} Q(z) \, dz = 0$$

nach r auf und bestimmt daraus die Größe des linken Teilintervalls l in Abhängigkeit von der Größe des rechten Teilintervalls r. Daraus folgt, daß man ohne große Einschränkung mit einem symmetrischen Intervall um die Nullstelle des Polynoms weiterrechnen kann. Dort gilt für ein Polynom mit optimal gewählten Koeffizienten für eine Siebgrenze M_2 :

$$\int_{\frac{-M_2}{\sqrt{2}}-\frac{w}{2}}^{\frac{-M_2}{\sqrt{2}}+\frac{w}{2}} |Q_2(z)|\ dz = \frac{\sqrt{N}\cdot w^2}{4}\ .$$

Da man um beide Nullstellen des Polynoms sieben will, muß man diesen Wert noch verdoppeln bevor man ihn mit dem Wert (4.3) vergleichen kann. Um beide Alternativen für gleich große Siebintervallängen ($w = M_1$) zu vergleichen, betrachtet man die Differenz beider Werte.

Es gilt :

$$\begin{aligned}
\int_{-M_1}^{M_1} |Q_1(z)|\ dz \quad - \quad 2\cdot \int_{\frac{-M_2}{\sqrt{2}}-\frac{M_1}{2}}^{\frac{-M_2}{\sqrt{2}}+\frac{M_1}{2}} |Q_2(z)|\ dz \quad &= \\
\frac{\sqrt{N}\cdot M_1^2}{3\cdot\sqrt{2}}\cdot(\sqrt{8}-1) - \frac{\sqrt{N}\cdot M_1^2}{2} \quad &= \\
\sqrt{N}\cdot M_1^2\left(\frac{\sqrt{8}-1}{\sqrt{18}}-\frac{1}{2}\right) \quad &< \quad 0\ .
\end{aligned}$$

Daraus folgt :

$$\int_{-M_1}^{M_1} |Q_1(z)|\ dz \quad < \quad 2\cdot \int_{\frac{-M_2}{\sqrt{2}}-\frac{M_1}{2}}^{\frac{-M_2}{\sqrt{2}}+\frac{M_1}{2}} |Q_2(z)|\ dz$$

Schließt man von der Größe der Beträge der Funktionswerte auf die Wahrscheinlichkeit des Findens von *full relations* und berücksichtigt man den zusätzlichen algorithmischen Aufwand, so erscheint die Wahl des kleineren Siebintervalls als günstiger. In Abbildung 4.5 sind die Durchschnittswerte der ρ-Funktion in Teilintervallen des Siebbereichs für die Siebgrenze $M = 10^6$ und $M = 3\cdot 10^6$ eingezeichnet.

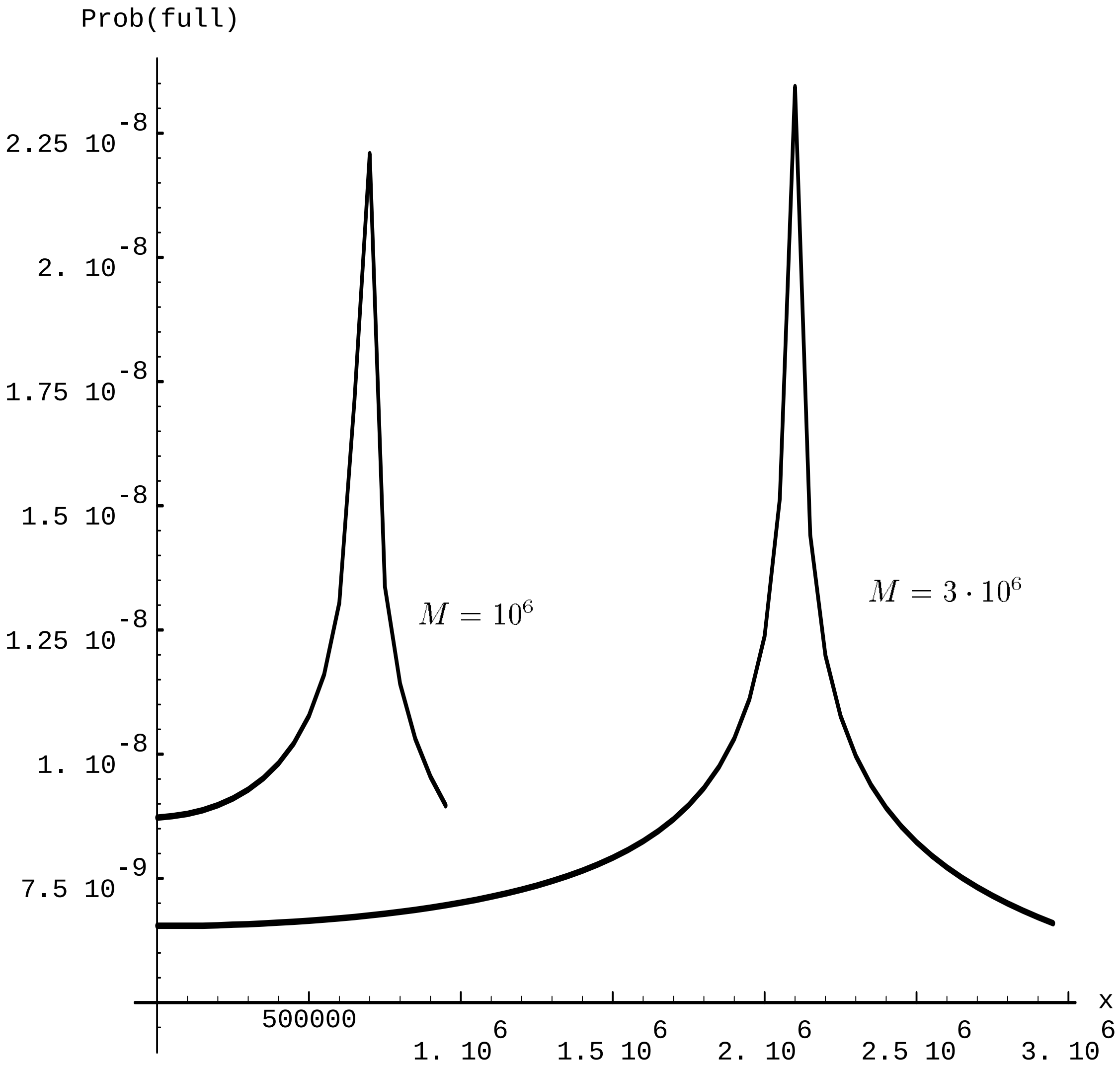


Abbildung 4.5: Vergleich der ρ Werte für zwei Siebintervallängen

4.2.4 Laufzeitauswirkungen

Durch die *Large Prime Variante (LPV)* wird der Testwert, mit dem jeder Siebeintrag nach dem Sieben verglichen wird, erniedrigt. Dadurch wird etwas mehr Zeit in der Auswertephase des Basisalgorithmus verbraucht. Man benötigt dadurch mit der *Large Prime Variante* etwa 10 % mehr Zeit, um eine *full relation* zu finden, als mit der Basisversion. Dies führt anfänglich zu einem kleinen Nachteil, der jedoch mit zunehmender Laufzeit der Faktorisierung mehr als ausgeglichen wird. Im allgemeinen kann man sagen, daß die *Large Prime Variante* zu einer Halbierung der Laufzeit führt. In Abbildung 4.6 ist das typische Beispiel für den Verlauf einer Faktorisierung mit der *Large Prime Variante* angegeben. Je nach Größe der Zahl und Wahl der Parameter verschieben sich die Graphen leicht. Die Abbildung enthält die Anzahl der *full relations*, die aus *partial relations* gebildet werden können, und gestrichelt die Anzahl der *full relations*, die in der selben Zeit gefunden wurden.

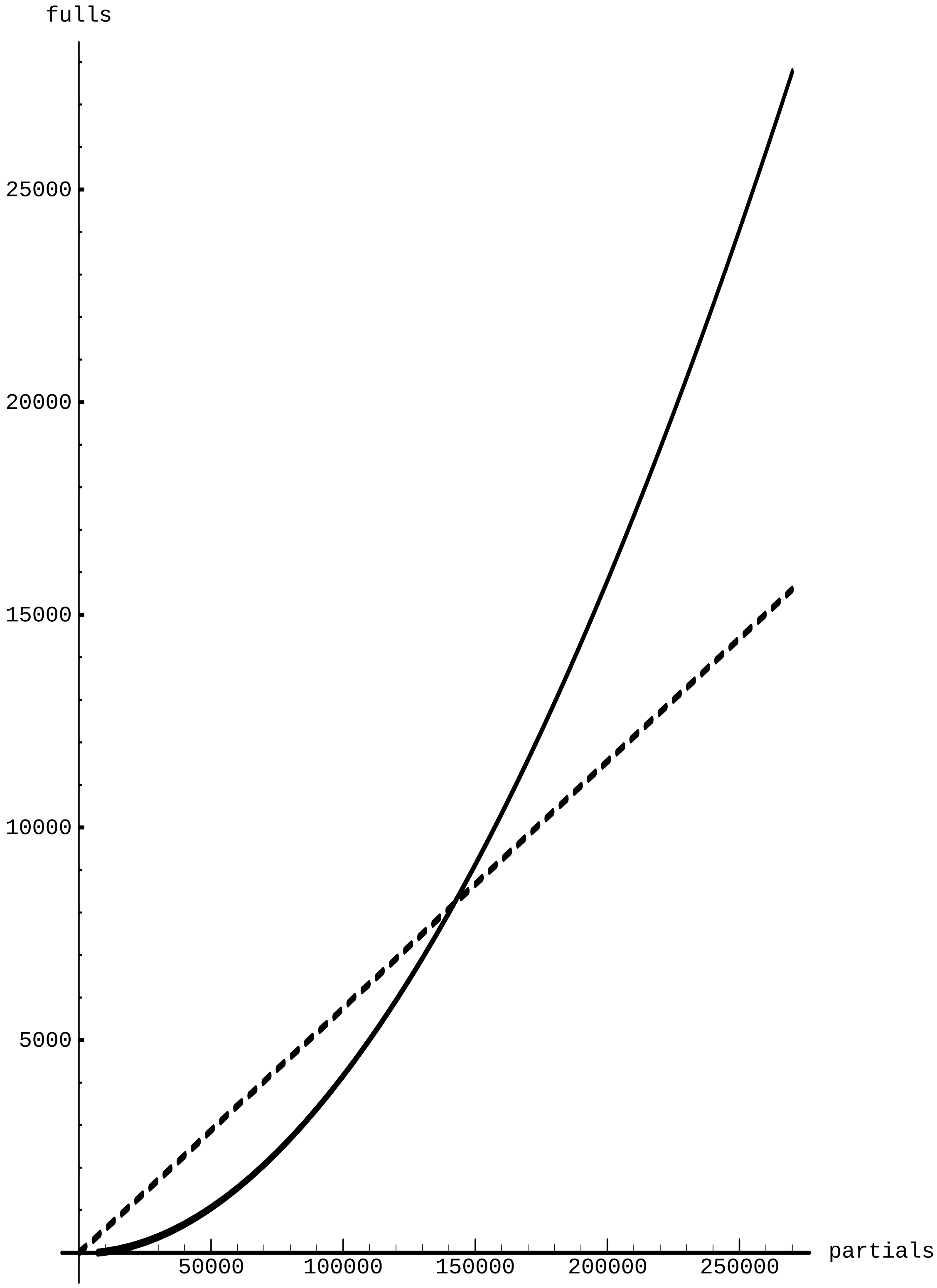


Abbildung 4.6: Typische Verlauf von Faktorisierungen mit der *LPV*

4.3 Double Large Prime Variante

Die *Double Large Prime Variante* ist die konsequente Fortsetzung und Erweiterung der *Large Prime Variante*. Man nutzt nun zusätzlich auch Zerlegungen von Funktionswerten, die 2 *large primes* beinhalten (vgl. 4.2) zur Konstruktion von *full relations*.
Diese Zerlegungen haben folgende Form :

$$Q(z) = q_1 \cdot q_2 \cdot \prod_{p_i \in FB} p_i^{e_i}$$

mit

$$p_{max} = max\{p \in FB\} < q_1 \leq q_2 < LARGE < p_{max}^2$$

Die dazugehörigen Relationen bezeichnet man als *partial-partial relations*. Diese haben folgende Form :

$$X^2 \equiv q_1 \cdot q_2 \cdot \prod_{p_i \in FB} p_i^{e_i} \bmod N \ . \tag{4.4}$$

Da man *partial* und *partial-partial relations* gemeinsam benutzt, um *full relations* zu bilden, werde ich im folgenden eine *partial relation*

$$X^2 \equiv q \cdot \prod_{p_i \in FB} p_i^{e_i} \bmod N$$

als *partial-partial relation* der Form (4.4) mit $q_1 = 1$ und $q_2 = q$

$$X^2 \equiv 1 \cdot q \cdot \prod_{p_i \in FB} p_i^{e_i} \bmod N$$

auffassen und diese Relationen generell als *partial relations* (im Gegensatz zu *full relations* der Form (4.1)) bezeichnen, falls keine genauere Angabe des Typs erforderlich ist.

Um aus *partial relations* eine *full relation* zu kombinieren, sucht man eine Teilmenge T dieser *partial relations*, die man aufmultipliziert. Diese Teilmenge T wird so gewählt, daß alle *large primes*, die in der Zerlegung der aufmultiplizierten rechten Seiten vorkommen, gerade Exponenten haben. Sei pr_j die j-te *partial relation* in T

$$pr_j : \quad X_j^2 \equiv q_{j,1} \cdot q_{j,2} \cdot \prod_{p_i \in FB} p_i^{e_{i,j}} \bmod N \ .$$

Dann bezeichnet Q_T die Menge der *large primes*, die in *partial relations* aus T vorkommen.

$$Q_T := \{q \mid q = q_{j,1} \ \vee \ q = q_{j,2} \quad \text{für } j \in [1\,,\, |T|\,]\}\ .$$

Dann gilt:

$$\prod_{j=1}^{|T|} X_j^2 \equiv \left(\prod_{k=1}^{|Q_T|} q_k^{\gamma_k}\right) \cdot \left(\prod_{p_i \in FB} p_i^{\sum_{j=1}^{|T|} e_{i,j}}\right) \bmod N \qquad \text{mit} \qquad \underset{k=1,\ldots,|Q_T|}{\forall} \quad \gamma_k \equiv 0 \bmod 2$$

und man hat eine *full relation* konstruiert.

Man könnte die Bestimmung der Teilmengen T auf die Lösungen eines linearen Gleichungssystems modulo $\mathbb{F}_2$ zurückführen. In diesem Gleichungssystem korrespondiert jede *partial relation* zu einer Spalte mit Einsern in den Zeilen, die zu ihren *large primes* gehören. Dies führt zu sehr großen Gleichnungssystemen, die nicht gelöst werden können. Außerdem muß man das Gleichungssystem immer lösen, um die Anzahl der *full relations* zu ermitteln, die aus den *partial relations* gebildet werden. Man benötigt diese Anzahl, um zu entscheiden, ob man genügend *full relations* gesammelt hat. Da es bisher keine genauen Abschätzungen für die Anzahl der *full relations*, die mit Hilfe der *Double Large Prime Variante* konstruiert werden können, gibt (vergleichbar mit denen für die *Large Prime Variante* (vgl. Kapitel 4.2.2)), stellt das Lösen des Gleichungssystems die einzige Möglichkeit zur Ermittelung dieser Anzahl dar.

Die im folgenden vorgestellte Methode zum Konstruieren der *full relations* hat den Vorteil, daß man auch sehr schnell die Anzahl der konstruierten *full relations* bestimmen kann (ohne diese zusammenzusetzen)([LenMan]). Es handelt sich dabei um einen Graphenalgorithmus auf Basis des Union-Find Problems ([MehTsa]).

Sei $G = (\ Q,\ V)$ ein gerichteter Graph und REL die Menge aller *partial relations*. Die Menge der Knoten Q des Graphen sei die Menge der in allen *partial relations* vorkommenden *large primes*.

$$Q := \{q \mid q = q_{j,1} \ \vee \ q = q_{j,2} \quad \text{für } j \in [1\,,\, |REL|\,]\}$$

Die Menge der Kanten V wird durch die *partial relations* gebildet. Es existiert eine Kante zwischen den Knoten q_2 und q_1, falls es eine *partial relation* mit q_1 und q_2 als *large primes* gibt. Es gilt :

$$(q_2, q_1) \in V \iff \underset{pr_j}{\exists} \ q_{j,1} = q_1 \ \wedge \ q_{j,2} = q_2\ .$$

Durchläuft man einen Zykel und multipliziert die entsprechenden *partial relations* auf, so gibt es für jede *large prime* eine Kante, die zu ihrem Knoten führt, und eine Kante, die wieder weggeht. Jede *large prime*, deren Knoten auf den Zykel liegt, erscheint deshalb in der aufmultiplizierten Kongruenz in gerader Anzahl. Deshalb korrespondiert jeder Zykel im Graph G zu einer *full relation*. Um linear abhängige *full relations* zu vermeiden, ist man nur an einer Basis der Zykel des Graphen interessiert.

4.3.1 Zählen der gebildeten *full relations*

Der Algorithmus zum Zählen der gebildeten *full relations* nutzt aus, daß die Anzahl der fundamentalen Zykel eines Graphen (Anz) aus der Anzahl der Kanten (e), der Knoten (v) und der Zusammenhangskomponenten (c) ermittelt werden kann (vgl. [LenMan, S.5]). Es gilt :

$$Anz = e + c - v \ .$$

Der Algorithmus hat als Eingabe die Menge REL der *partial relations*. Er sortiert die doppelten Relationen aus, und testet, ob es sich um korrekte Relationen handelt, also ob die Kongruenz

$$X^2 \equiv q_1 \cdot q_2 \cdot \prod_{p_i \in FB} p_i^{e_i} \bmod N$$

erfüllt ist. Diese Vorberechnungen sind notwendig, da durch Booten von Maschinen oder Übertragungsfehler *partial relations* teilweise zerstört oder doppelt aufgenommen werden können. Danach kann aus der Anzahl der übrigen Relationen die Anzahl der Kanten (e) bestimmt werden. Es ist jedoch einfacher den Wert von e bei der Bestimmung von c und v mitzubestimmen.

Um eine Repräsentation des Graphen bzw. seiner Zusammenhangskomponenten abzuspeichern, benötigt man eine Hashtafel $T = (d_i, a_i)_{i=0}^{S-1}$. Die 1. Komponente der Hashtafel (d_i) nimmt die *large primes* auf. Sie enthält entweder den Wert -1 (Hasheintrag frei) oder die *large prime*, die beim Hashwert i abgespeichert wurde. Die 2. Komponente (a_i) enthält einen Verweis auf die Wurzel der Komponente, in der die *large prime* des Hasheintrags steht. Pro Hasheintrag werden je 4 Bytes für d_i und a_i benötigt. Die Hashtafel sollte aus Effizienzgründen möglichst groß gewählt werden, jedoch noch ganz im Hauptspeicher gehalten werden können. Die Wahl von $S = 2{\cdot}10^6$ hat sich in der Praxis bewährt. Außerdem benötigt man eine Hashfunktion

$$f : \mathbb{N} \rightarrow [\ 0 \ldots S-1\ ]\ .$$

Eine mögliche Wahl stellt

$$f(x) = x \bmod S$$

dar. Es wird nicht der komplette Graph mit allen Kanten aufgebaut, sondern man ermittelt nur die Zusammenhangskomponenten des Graphen. Dazu werden nur diejenigen Kanten aufgenommen, die nicht in derselben Zusammenhangskomponente liegen. Diese Kanten verbinden zwei verschiedene Zusammenhangskomponenten. Man aktualisiert dabei die Anzahl der Kanten, Knoten und Zusammenhangskomponenten.

Nachdem e *partial relations* in den Graph aufgenommen wurden, gilt immer für die Anzahl (Anz) der fundamentalen Zykel im Graph :

$$Anz = e + c - v \ .$$

In der Darstellung des Algorithmus verwende ich einige Funktionen, die ich kurz erläutern möchte.

suche(q) : Die Funktion ermittelt den Hashwert $f(q)$ und sucht ab $f(q)$ zyklisch in der Hashtabelle den Eintrag für q oder die nächste freie Stelle der Hashtafel. Sie liefert den Index der gefundenen Stelle zurück.

$j \leftarrow q \bmod S$;			(1)
WHILE ((d_j != -1) && (d_j != q))			(2)
	IF ($j < S-1$)		(3)
	THEN	$j \leftarrow j+1$,	(4)
	ELSE	$j \leftarrow 0$,	(5)
Return j ;			(6)

einfüge_z(q) : Die Funktion fügt einen Eintrag für die *large prime* q in die Hashtafel ein, falls dieser nicht bereits existiert. Sie aktualisiert gegebenenfalls die Anzahl der Knoten und Komponenten.

$j \leftarrow suche(q)$;		(1)
IF (d_j == -1)		(2)
THEN	$d_j \leftarrow q$;	(3)
	$a_j \leftarrow j$;	(4)
	$v++$;	(5)
	$c++$;	(6)

komp(q) : Die Funktion ermittelt die Wurzel der Zusammenhangskomponente in der q liegt. Die Wurzel einer Zusammenhangskomponente zeigt auf sich selbst. (Zur Effizienzsteigerung setzt die implementierte Funktion alle Verweise auf dem Suchpfad zur Wurzel auf die Wurzel.)

$r \leftarrow suche(q)$;		(1)
WHILE $((a_r \mathrel{!=} r)$		(2)
	$r \leftarrow a_r$;	(3)
Return r ;		(4)

Der Algorithmus, der aus der Menge der *partial relations*, die Anzahl der daraus gebildeten *full relations* ermittelt, sieht folgendermaßen aus:

Algorithmus 4.1 (Zählen der Zykel)

Eingabe: Datei mit den *partial relations*

Ausgabe: Anzahl der Zykel bzw. der konstruierbaren *full relations*

FOR $i = 0$ TO $S - 1$ /* Initialisieren der Hashtafel */				(1)
	$d_i \leftarrow -1$;			(2)
$v \leftarrow 0$; $e \leftarrow 0$; $c \leftarrow 0$;				(3)
WHILE (noch nicht alle Relationen verarbeitet)				(4)
	Lese neue Relation mit *large primes* q_1, q_2 ein ; $e++$;			(5)
	$einfüge\_z(q_1)$; $einfüge\_z(q_2)$;			(6)
	IF $((r_1 = komp(q_1)) \mathrel{!=} (r_2 = komp(q_2)))$			(7)
	THEN	$c = c - 1$;		(8)
		IF $(d_{r_1} < d_{r_2})$		(9)
		THEN	$(a_{r_2} \leftarrow r_1)$	(10)
		ELSE	$(a_{r_1} \leftarrow r_2)$	(11)
Return $(e + c - v)$				(12)

In Schritt (7) wird getestet, ob die beiden *large primes* zu verschiedenen Zusammenhangskomponenten gehören. Ist dies der Fall, so verbindet die gerade betrachtete *partial relation* zwei Zusammenhangskomponenten und deren Anzahl wird um eins erniedrigt (Schritt (8)). Dann werden die beiden Zusammenhangskomponenten verbunden, indem der Verweis der Wurzel mit der größeren *large prime* auf die Wurzel mit der kleineren *large prime* gesetzt wird. Aus statistischen Gründen zähle ich die Anzahl, der von *partial* und *partial-partial relations* erzeugten *full relations*, getrennt.

4.3.2 Konstruieren der *full relations*

Ist die Anzahl der gebildeten und normalen *full relations* größer als die Anzahl der Faktorbasiselemente, so kann man die Phase des Sammelns der Relationen abbrechen. Man hat genügend *full relations* gesammelt, um das Gleichungssystem (2.5) zu lösen. Erst jetzt konstruiert man die *full relations*, die aus den *partial relations* gebildet werden können.

In der Praxis hat es sich als äußert nützlich erwiesen, zuerst diejenigen *partial relations* auszusortieren, die *large primes* enthalten, die nur einmal in allen *partial relations* vorkommen. Diese können nicht an der Bildung eines Zykels im Graph beteiligt sein. Man ermittelt zuerst für jede *large prime* die Häufigkeit ihres Auftretens in allen *partial relations*. Dazu benutzt man fast die gleiche Hashtafel wie zum Zählen der Zykel. Der Unterschied besteht in der 2. Komponente. Sie wird zum Zählen des Vorkommens der jeweiligen *large prime* benutzt. Ihr Platzbedarf reduziert sich auf ein Byte, so daß die Hashtafel mehr Einträge haben kann. Man liest alle *partial relations* ein und aktualisiert die Zähler für die in ihnen enthaltenen *large primes*, die größer als 1 sind.

Hat man alle Relationen eingelesen, so liest man sie zum 2. Mal ein, und entfernt diejenigen Relationen, die eine *large prime* größer als 1 enthalten, deren Zähler gleich 1 ist. Entfernt man eine Relation, so erniedrigt man auch die Zähler, der in ihr enthaltenen *large primes*. Das Entfernen von *partial-partial relations* kann Seiteneffekte bezüglich anderer Relationen hervorrufen, so daß es sich empfiehlt, diesen Schritt öfter zu wiederholen. In der Praxis wird dieser Schritt zweimal durchlaufen. Danach hat sich die Anzahl der in Frage kommenden *partial relations* schon sehr stark verringert, wie das folgende Beispiel verdeutlicht.

Um eine 94-stellige Zahl zu faktorisieren (Kofaktor von p(20021)), wurden 413 002 *partial-partial* und 154 435 *partial relations*, also insgesamt 567 437 *partial relations*, gesammelt. An der Konstruktion der 30 389 *full relations* waren 85 139 *partial relations* beteiligt. Nach dem ersten Eliminationsschritt waren von den insgesamt 567 437 gesammelten *partial relations* noch 171 439 übrig. Der zweite Durchlauf führte zu einer weiteren Reduzierung auf 107 612 Relationen, was nur noch etwa 18 % der ursprünglich gesammelten Relationen ausmacht.

Algorithmus 4.2 (Entfernen der unbrauchbaren Relationen)

Eingabe: Datei mit den *partial relations*

Ausgabe: verkürzte Datei (ohne die sicher unbrauchbaren Relationen)

FOR $i = 0$ TO $S-1$ /∗ Initialisieren der Hashtafel ∗/			(1)
	$d_i \leftarrow -1$; $a_i \leftarrow 0$;		(2)
/∗ Aufbauen der Hashtafel und Zählen der Häufigkeit ∗/			(3)
WHILE (noch nicht alle Relationen eingelesen)			(4)
	Lese neue Relation mit *large primes* q_1, q_2 ein ;		(5)
	IF (($j_1 \leftarrow suche(q_1)$) $== -1$)		(6)
	THEN	$d_{j_1} \leftarrow q_1$;	(7)
	IF (($j_2 \leftarrow suche(q_2)$) $== -1$)		(8)
	THEN	$d_{j_2} \leftarrow q_2$;	(9)
	$a_{j_1}{+}{+}$; $a_{j_2}{+}{+}$;		(10)
/∗ Löschen von sicher unproduktiven Relationen ∗/			(11)
WHILE (noch nicht alle Relationen erneut eingelesen)			(12)
	Lese neue Relation mit *large primes* q_1, q_2 ein ;		(13)
	$j_1 \leftarrow suche(q_1)$; $j_2 \leftarrow suche(q_2)$;		(14)
	IF (($a_{j_1} == 1$) || ($a_{j_2} == 1$))		(15)
	THEN	$a_{j_1}{-}{-}$; $a_{j_2}{-}{-}$;	(16)
		Lösche Relation ;	(17)

Dabei kann die letzte Phase (Schritt (12) - (17)) öfter durchlaufen werden. Es ist sinnvoll, zuerst die *partial-partial relations* zu lesen und dann erst die *partial relations*, da so bereits im 1. Durchlauf die Auswirkungen der gelöschten *partial-partial relations* verarbeitet werden. Durch diese Vorberechnungen wurde die Anzahl der zu bearbeitenden Relationen stark verkürzt. Dies ist aus Platzgründen für das eigentliche Konstruieren der *full relations* wichtig.

Die Basis für den Algorithmus zum Konstruieren der *full relations* stellt der Algorithmus zum Zählen der gebildeten *full relations* dar (vgl. Algorithmus 4.1). Die 2. Komponente der Hashtafel zeigt nun auf den direkten Vorgänger eines Knotens auf dem Weg zur Wurzel. Außerdem enthält die Hashtafel eine 3. Komponente (r_i) mit einem Pointer auf die Position, an der die zugehörige Relation, in der Datei abgespeichert ist. Dabei wird diese Information für die Kante (q_1, q_2) am Knoten q_1 abgespeichert. Es wird wiederum nicht der ganze Graph aufgebaut. Dazu müßten für einen Knoten bzw. beim Hasheintrag des Knotens mehrere Kanten verwaltet werden, was eine platzaufwendige Datenstruktur (verkettete Liste an jedem Knoten) zur Folge hätte. Der Algorithmus baut einen aufspannenden Baum der Zusammenhangskomponenten auf. Wird ein Zykel gefunden, so wird die ihn schließende Relation nicht aufgenommen, sondern die dazugehörige *full relation* berechnet und abgespeichert. Der Algorithmus findet nicht die kürzesten Zykel, was Vorteile beim Lösen des Gleichungssystems (2.5) hätte. Es existieren zwar auch Graphenalgorithmen, die die kürzesten Zykel eines Graphen finden, deren Laufzeit aber zu hoch ist. Im vorgestellten Algorithmus ist es ein Vorteil, zuerst die *partial relations* und dann erst die *partial-partial relations* einzulesen. Dies führt zu wesentlich kürzeren Zykeln. Zur Beschreibung des Algorithmus benutze ich, zusätzlich zu den bereits definierten, folgende Funktionen :

einfüge_k(q) : Die Funktion fügt einen Eintrag für die *large prime* q in die Hashtafel ein, falls dieser nicht bereits existiert.

$j \leftarrow suche(q)$;		(1)
IF $(d_j\ ==\ -1)$		(2)
THEN	$d_j \leftarrow q$; $a_j \leftarrow j$;	(3)

reverse(i,j) : Die Funktion läuft den Pfad von Knoten i zum Knoten j ab, und vertauscht dabei die Orientierung der Kanten. Außerdem wird der Verweis auf die Relationen gepflegt.

$vorgänger \leftarrow i$; $aktuell \leftarrow a_i$; $nachfolger \leftarrow a_{aktuell}$;		(1)
$rel \leftarrow r_{aktuell}$; $a_{aktuell} \leftarrow vorgänger$; $r_{aktuell} \leftarrow r_{vorgänger}$;		(2)
WHILE ($aktuell\ != j$)		(3)
	$vorgänger \leftarrow aktuell$; $aktuell \leftarrow nachfolger$;	(4)
	$nachfolger \leftarrow a_{aktuell}$; $a_{aktuell} \leftarrow vorgänger$;	(5)
	$hilfrel \leftarrow r_{aktuell}$; $r_{aktuell} \leftarrow rel$; $rel \leftarrow hilfrel$;	(6)

Algorithmus 4.3 (Konstruieren der *full relations*)

Eingabe: verkürzte Datei mit den *partial relations*

Ausgabe: konstruierbare *full relations*

FOR $i = 0$ TO $S - 1$ /* Initialisieren der Hashtafel */					(1)
				$d_i \leftarrow -1$; $a_i \leftarrow 0$; $r_i \leftarrow 0$;	(2)
WHILE (noch nicht alle Relationen verarbeitet)					(3)
				Lese neue Relation mit *large primes* q_1, q_2 ein ;	(4)
				$einfüge\_k(q_1)$; $einfüge\_k(q_2)$;	(5)
	IF ($(r_1 = komp(q_1))$) != ($r_2 = komp(q_2))$)				(6)
	THEN	IF ($d_{r_1} < d_{r_2}$)			(7)
		THEN		$max = 2$;	(8)
		ELSE		$max = 1$;	(9)
		IF (r_{max} != j_{max})			(10)
		THEN		reverse(j_{max} , r_{max}) ;	(11)
				$a_{j_{max}} \leftarrow j_{(3-max)}$; $r_{j_{max}} \leftarrow$ aktueller Filepointer ;	(12)
	ELSE			reverse(j_1 , r_2) ; $flag \leftarrow 0$; $start \leftarrow j_2$;	(13)
				fullrel $\leftarrow$ aktuelle *partial relation* ;	(14)
		WHILE ($start$!= j_1)			(15)
			IF ($flag$)		(16)
			THEN	fullrel $\leftarrow$ fullrel $\cdot$ partrelation(r_{start}) ;	(17)
			ELSE	fullrel $\leftarrow$ fullrel / partrelation(r_{start}) ;	(18)
				$flag \leftarrow !flag$; $start \leftarrow a_{start}$;	(19)
				reverse(r_2 , j_1) ; $a_{r_1} \leftarrow r_1$;	(20)
				Ausgabe der neuen *full relation* ;	(21)

In Schritt (7) - (12) werden zwei verschiedene Zusammenhangskomponenten vereinigt, die durch die aktuell betrachtete Kante verbunden werden. Dabei wird die Zusammenhangskomponente mit der größeren *large prime* r_{max} als Wurzel mit der anderen Zusammenhangskomponente verbunden. Dadurch wird in der Regel die kleinere Zusammenhangskomponente an die größere Zusammenhangskomponente angehängt. Man vertauscht die Orientierung der Kanten auf dem Pfad von j_{max} nach r_{max} und nimmt die Kante zwischen j_{max} und $j_{(3-max)}$ auf.
In Schritt (13) - (21) wird die neu gebildete *full relation* zusammengebaut. Dabei wird zuerst die Orientierung der Kanten auf dem Pfad von j_1, des Knotens, der zur kleineren *large prime* der zykelschließenden Relation gehört, zu r_2, der Wurzel der Zusammenhangskomponente vertauscht (vgl. **reverse**). Dadurch ist es möglich den kompletten Zykel von j_2 nach j_1 zu durchlaufen. Man beginnt bei der Konstruktion der neuen *full relation* mit der aktuell betrachteten *partial relation*

$$X_1^2 \equiv q_1 \cdot q_2 \cdot \prod_{p_i \in FB} p_i^{e_{i,1}} \bmod N \ .$$

Man durchläuft nun den ganzen Zykel und dividiert bzw. multipliziert abwechselnd das bisherige Ergebnis mit der zur gerade durchlaufenen Kante gehörenden *partial relation*. Sei

$$X_2^2 \equiv q_2 \cdot q_3 \cdot \prod_{p_i \in FB} p_i^{e_{i,2}} \bmod N \ .$$

die *partial relation*, die zur ersten Kante gehört. Nach der Divison erhält man folgende Kongruenz :

$$X_1^2 \cdot q_3 \equiv X_2^2 \cdot q_1 \cdot \prod_{p_i \in FB} p_i^{(e_{i,1}-e_{i,2})} \bmod N \ . \tag{4.5}$$

Die zur nächsten Kante gehörende *partial relation*

$$X_3^2 \equiv q_3 \cdot q_4 \cdot \prod_{p_i \in FB} p_i^{e_{i,3}} \bmod N$$

wird mit der bisher gefundenen Kongruenz multipliziert und man erhält:

$$(X_1 \cdot X_3)^2 \equiv X_2^2 \cdot q_1 \cdot q_4 \cdot \prod_{p_i \in FB} p_i^{(e_{i,1}-e_{i,2}+e_{i,3})} \bmod N \ . \tag{4.6}$$

Man wiederholt diesen Vorgang solange, bis der Zykel ganz abgearbeitet wurde.
Sei

$$X_j^2 \equiv q_{j,1} \cdot q_{j,2} \cdot \prod_{p_i \in FB} p_i^{e_{i,j}} \bmod N$$

die zur j-ten betrachteten Kante auf dem Zykel gehörenden *partial relation*. Hat der gefundene Zykel gerade Länge (z.B. $2 \cdot n$), so erhält man an Ende eine Kongruenz der Form (4.5) mit $q_1 = q_3$ und man erhält die neue *full relation*

$$X_U^2 \cdot \left(X_G^2\right)^{-1} \equiv \prod_{p_i \in FB} p_i^{e_{i,g}} \bmod N$$

mit

$$X_U = \prod_{k=0}^{n-1} X_{2\cdot k+1} \ , \quad X_G = \prod_{k=1}^{n} X_{2\cdot k} \ , \quad e_{i,g} = \sum_{k=0}^{n-1} e_{i,2\cdot k+1} - \sum_{k=1}^{n} e_{i,2\cdot k} \ .$$

Ist die Zykellänge ungerade (z.B. $2 \cdot n + 1$), so erhält man eine Kongruenz der Form (4.6) mit $q_1 = q_4$ und daraus eine neue *full relation*

$$X_U^2 \cdot \left((X_G \cdot q_1)^2\right)^{-1} \equiv \prod_{p_i \in FB} p_i^{e_{i,g}} \bmod N$$

mit

$$X_U = \prod_{k=0}^{n} X_{2\cdot k+1} \ , \quad X_G = \prod_{k=1}^{n} X_{2\cdot k} \ , \quad e_{i,g} = \sum_{k=0}^{n} e_{i,2\cdot k+1} - \sum_{k=1}^{n} e_{i,2\cdot k} \ .$$

Dabei wird in beiden Fällen die Inversenberechnung nicht durchgeführt, sondern man speichert die neuen *full relations* in der Form

$$X^2 \equiv I^2 \equiv \prod_{p_i \in FB} p_i^{e_i} \bmod N \ ,$$

wobei im Fall der geraden Zykellänge gilt :

$$X^2 \equiv X_U^2 \bmod N \ , \quad I^2 \equiv X_G^2 \bmod N \ , \quad \text{und } e_i = e_{i,g} \ .$$

Für ungerade Zykellänge gilt :

$$X^2 \equiv X_U^2 \bmod N \ , \quad I^2 \equiv (X_G \cdot q_1)^2 \bmod N \ , \quad \text{und } e_i = e_{i,g} \ .$$

Wird diese neue *full relation* zum Aufbau der quadratischen Kongruenz (vgl. (2.1) und Kapitel 3.7) benötigt, so wird I beim Aufbau von Y berücksichtigt.

4.3.3 Finden von *partial-partial relations*

Im Gegensatz zur *Large Prime Variante* genügt es bei der *Double Large Prime Variante* nicht, den Testwert der Auswertung der Siebphase (vgl. Kapitel 3.6) zu erniedrigen, um *partial-partial relations* zu finden. Die Auswertung der Siebphase muß zusätzlich noch leicht modifiziert werden. Durch die Herabsetzung des Testwertes erhält man größere Reste R der Funktionswerte, die nicht über der Faktorbasis zerfallen. Untersucht man in der Testphase einen Funktionswert im Siebintervall genauer, so erhält man, nach Ermittelung der Faktorbasiselemente, die ihn teilen, folgende Kongruenz

$$X^2 \equiv R \cdot \prod_{p_i \in FB} p_i^{e_i} \bmod N \ .$$

Ist dieser Rest R gleich eins, so hat man eine *full relation* gefunden.

Gilt

$$p_{max} := max\{p \in FB\} < R < LARGE$$

so hat man eine *partial relation* gefunden (vgl. Kapitel 4.2.1).

Eine notwendige Bedingung zum Finden einer *partial-partial relation* ist :

$$p_{max}^2 \leq R \leq LARGE^2 \ .$$

Erfüllt R diese Bedingung, so wird ein Primzahltest mit R durchgeführt. Erkennt dieser, daß R zusammengesetzt ist, so faktorisiert man R und erhält zwei Teiler q_1 und q_2 (mit $q_1 \leq q_2$). Dies gilt, da R, falls R zusammengesetzt ist und $LARGE^2 < p_{max}^3$ ist, genau 2 Teiler hat. Dabei benutzt man, daß R keine Teiler aus der Faktorbasis hat. Sind beide Teiler kleiner als $LARGE$, so hat man eine *partial-partial relation*

$$X^2 \equiv q_1 \cdot q_2 \cdot \prod_{p_i \in FB} p_i^{e_i} \bmod N \qquad \text{gefunden.}$$

Für eine *partial-partial relation* mit gleichgroßen *large primes* nahe bei LARGE ist die Wahrscheinlichkeit, an der Konstruktion einer *full relation* beteiligt zu sein, relativ klein. Deshalb wird als obere Grenze für R nicht $LARGE^2$ sondern ein kleinerer Wert z.B. $0.5 \cdot LARGE^2$ gewählt.

Die Effizienz der*Double Large Prime Variante* hängt stark von der Effizienz des zur Zerlegung von R benutzten Faktorisierungsalgorithmus ab. Dabei handelt es sich um ein spezielles Faktorisierungsproblem, da man genau weiß, daß die Faktoren von R eine spezielle Form haben und daß R genau 2 Teiler hat. Außerdem sind die zu faktorisierenden Zahlen R kleiner als 10^{20}. Zur Lösung dieses speziellen Faktorisierungsproblems eignen sich Pollard's ρ-Methode (vgl. [Rie, S.174], [Bre, S.62]) und der Square Forms Factorization (SQUFOF) Algorithmus von D. Shanks (vgl. [Rie, S.191],[Coh, S.292]).

In der Praxis erwies sich SQUFOF als der schnellere Algorithmus. Aus Effizienzgründen läßt man die Algorithmen nicht durchlaufen, bis sie die Teiler ermittelt haben, sondern beschränkt die Anzahl der Iterationen. Dadurch werden die Algorithmen schneller, die Anzahl der Fehlversuche, bei denen innerhalb der zugelassenen Iterationen kein Teiler gefunden wurde, nimmt jedoch zu. Die Anzahl der Fehlversuche liegt beim SQUFOF jedoch nur bei ungefähr 1%. Da es außerdem Zahlen gibt, die mit Pollard ρ schneller als mit SQUFOF faktorisiert werden können, hat sich in der Praxis ein zweistufiges Konzept bewährt. In der ersten Stufe versucht man mit einer angemessenen, aber nicht zu großen Anzahl von Iterationen ($\approx 50\,000$), einen Teiler mit SQUFOF zu finden. Ist dies nicht möglich, so versucht man die Teiler mit Pollard ρ zu ermitteln. Gelingt dies in angemessener Zeit nicht, so verzichtet man aus Laufzeitgründen auf diese potentielle *partial-partial relation.*

4.3.4 Steuerung und Einsatzmöglichkeiten

Mit Hilfe des bereits öfter benutzten Testwertes läßt sich der Algorithmus steuern. Wie bereits in Kapitel 3.5 erwähnt, ist nach der Siebphase der Wert der Siebeinträge ein Maß für die Größe der aufmultiplizierten Teiler des Funktionswertes aus der Faktorbasis. Somit kann man durch Variieren dieses Testwertes die Größe des nicht über der Faktorbasis zerfallenden Restes R verändern. Man kann bestimmen, ob der Algorithmus in seiner Basisversion (Anheben des Testwertes, so daß fast nur *full relations* gefunden werden), mit der *Large Prime Variante* (Herabsenken des Testwertes, so daß auch zusätzlich noch *partial relations* gefunden werden, aber kaum *partial-partial relations*) oder mit *Double Large Prime Variante* (weiteres Absenken des Testwertes, so daß möglichst viele *partial-partial relations* gefunden werden) arbeiten soll.

Dieses Vorgehen hat sich in der Praxis als vorteilhaft gegenüber dem expliziten Verbieten von *Double* oder *Large Prime Variante* erwiesen. Mit dieser Strategie wird versucht, möglichst alle Informationen, die durch die Siebphase berechnet wurden, auszunutzen. Ein weiterer Vorteil dieser Strategie ist die Möglichkeit des gleitenden Überganges zwischen den einzelnen Varianten. Es muß nicht explizit eine Grenze angegeben werden, ab der es sich lohnt, die *Double Large Prime Variante* zu benutzen. Es ist prinzipiell nicht sinnvoll, ohne *Large Prime Variante* zu arbeiten.

Der Wahl des Testwertes kommt eine entscheidende Rolle bei der Effizienz des Algorithmus zu. Ein möglicher kleiner Nachteil dieser Strategie könnte in dem Fehlen der Möglichkeit, Faktorisierungen mit einer reiner Basisversion oder Basisversion mit *Large Prime Variante* durchzuführen, liegen. Dies liegt zum Teil auch daran, daß es durch die effizienzsteigernden Maßnahmen der Siebphase (Sieben ohne kleine Faktorbasiselemente und Potenzen von Faktorbasiselementen sowie das Arbeiten mit Näherungen von dualen Logarithmen) nicht möglich ist, für die Größe der Siebeinträge disjunkte Bereiche anzugeben, in denen nur *full*, *partial* oder *partial-partial relations* gefunden werden.

Dies wird durch die Tabelle 4.6 verdeutlicht. Sie listet für eine 90-stellige Zahl die untersuchten Siebeinträge bei 648 bearbeiteten Polynomen nach der Höhe ihres Testwertes auf. Dabei wird jeweils die Anzahl der untersuchten Stellen mit diesem Testwert, sowie die Anzahl der gefundenen *full*, *partial* und *partial-partial relations* angegeben. Es wurde eine Faktorbasis mit 37 000 Elementen und eine Siebintervallänge von $8 \cdot 10^6$ ($M = 4 \cdot 10^6$) gewählt. Der verwendete Multiplier war 3 und zur Bestimmung des Testwertes (vgl. (3.6)) wurde $T = 3.3$ benutzt.

Die Effizienz der *Double Large Prime Variante* nimmt (wie die Effizienz der *Large Prime Variante*) mit steigender Laufzeit der Faktorisierung (also steigender Größe der Zahlen) zu. Der Bereich, ab dem mit markanten Laufzeitverbesserungen durch den Einsatz der *Double Large Prime Variante* zu rechnen ist, hängt stark von der Implementierung ab. Wie auch in [LenMan] beschrieben, fängt dieser Bereich etwa bei zu faktorisierenden Zahlen von 80 Dezimalstellen an. Der Einsatz ist jedoch auch schon früher vorteilhaft, jedoch spielt dort die *Double Large Prime Variante* noch eine untergeordnete Rolle. Im nächsten Abschnitt werden einige Beispiele aufgelistet, die die zunehmende Bedeutung der *Double Large Prime Variante* in Abhängigkeit der Größe der zu faktorisierenden Zahlen veranschaulicht.

Testwert	# Stellen	# *fulls*	# *partials*	# *partial-partials*
128 - 129	3554	0	11	483
130 - 131	2734	0	9	440
132 - 133	2199	0	16	365
134 - 135	1572	0	14	294
136 - 137	1184	0	15	208
138 - 139	861	0	24	126
140 - 141	678	0	31	93
142 - 143	529	0	32	53
144 - 145	430	0	51	21
146 - 147	304	0	50	14
148 - 149	210	0	33	6
150 - 151	179	0	54	1
152 - 153	133	0	57	0
154 - 155	120	0	53	0
156 - 157	98	0	53	0
158 - 159	65	0	48	0
160 - 161	40	1	31	0
162 - 163	29	0	28	0
164 - 165	19	1	18	0
166 - 167	18	0	18	0
168 - 169	17	0	17	0
170 - 171	10	1	9	0
172 - 173	4	2	2	0
174 - 176	6	5	1	0
177 - 197	18	18	0	0

Tabelle 4.6: Einteilung der Siebeinträge nach ihrer Größe und gefundene Relationen

4.3.5 Laufzeitauswirkungen und praktische Erfahrungen

In diesem Abschnitt stelle ich einige praktische Erfahrungen bei Faktorisierungen mit der *Double Large Prime Variante* vor. Bei allen angegebenen Beispielen wurde $LARGE = 10^8$ gewählt. In der Spalte LP steht die Anzahl der *full relations*, die nur aus *partial relations* (also durch die *Large Prime Variante*) erzeugt wurden. Entsprechend enthält die Spalte DLP die Anzahl der *full relations*, die aus *partial* und *partial-partial relations* konstruiert werden können. In der letzten Spalte wird jeweils das Verhältnis der kombinierten zu den direkt gefundenen *full relations* aufgelistet. Diese Daten wurden an mehreren Zeitpunkten während der Faktorisierung ermittelt. Eine weitere Tabelle enthält die Verteilung der kombinierten *full relations* bezüglich der Länge der zugehörigen Zykel.

Das erstes Beispiel zeigt die untergeordnete Bedeutung der *Double Large Prime Variante* bei der Faktorisierung "kleinerer" Zahlen. Die zu faktorisierende Zahl ist eine 75-stellige Zahl (Kofaktor von p(14009)).

Beispiel für 75 Dezimalstellen

# fulls	# partials	LP	# partial-partials	DLP	DLP/ # fulls
3519	42046	1884	187	1902	0.54
4332	52904	2849	249	2876	0.66
7042	88663	7548	483	7670	1.09

Zykellänge

Länge	2	3	4
Anzahl	7548	120	2

Den wachsenden Einfluß der *Double Large Prime Variante* ist bereits bei der Faktorisierung einer 78-stelligen Zahl (Kofaktor von p(16061)) zu erkennen. Die Anzahl der gesammelten *partial-partial relations* steigt und damit auch die Anzahl der *full relations*, an deren Konstruktion *partial-partial relations* beteiligt sind. Außerdem werden die Zykel länger und die Anzahl der Zykel mit längeren Zykellängen nimmt zu.

Beispiel für 78 Dezimalstellen

# fulls	# partials	LP	# partial-partials	DLP	DLP/ # fulls
3830	49737	2097	15062	2442	0.64
7174	91737	6833	28024	8725	1.22

Zykellänge

Länge	2	3	4	5	6	7
Anzahl	6833	1521	306	56	8	1

Die *Double Large Prime Variante* liefert die meisten der benötigten *full relations*, falls die zu faktorisierenden Zahlen mehr als 85 Dezimalstellen haben. Die direkt gefundenen *full relations* verlieren immer stärker an Bedeutung. Dies verdeutlichen die Daten, die bei der Faktorisierung einer 89-stelligen Zahl (Kofaktor von p(15187)) gesammelt wurden.

Beispiel für 89 Dezimalstellen

# fulls	# partials	LP	# partial-partials	DLP	DLP/ # fulls
755	12456	75	33883	85	0.11
1704	28260	378	80647	495	0.29
3020	49811	1172	142485	1862	0.62
4355	71126	2373	205325	4696	1.08
5633	92407	4011	267776	9801	1.74
6568	107898	5363	311964	15076	2.30
7928	130608	7713	378725	26342	3.32

Zykellänge

Länge	Anzahl	Länge	Anzahl	Länge	Anzahl
2	7713	8	642	14	12
3	6813	9	375	15	9
4	4534	10	174	16	1
5	2990	11	99	17	1
6	1839	12	38	18	0
7	1074	13	27	19	1

Benutzt man den Algorithmus mit der *Double Large Prime Variante* so führt dies zur Wahl von kleineren Faktorbasen. Dadurch findet man weniger *full relations* direkt und die Bedeutung der *Double Large Prime Variante* nimmt zu. Während der kompletten Faktorisierung bleibt das Verhältnis der direkt gefundenen *full*, der *partial* und der *partial-partial relations* konstant. Diese Verhältnis wird durch die Wahl der Faktorbasisgröße beeinflußt.

Wichtig für die *Double Large Prime Variante* ist auch das Verhältnis der *partial-partial* zu den *partial relations*. Dieses kann durch den Testwert variiert werden. Da es bisher keine genaueren Untersuchungen der *Double Large Prime Variante* gibt, ist man auf praktische Erfahrungen angewiesen. Untersuchungen haben gezeigt, daß es kaum (ca. 0.008%) konstruierte *full relations* gibt, die nur aus *partial-partial relations* gebildet werden. Deshalb ist es keine gute Strategie, den Testwert so zu wählen, daß man möglichst viele *partial-partial relations* pro Zeit findet. Man sollte bei Optimierungsstrategien immer auch die Anzahl der *partial relations* pro Zeit und das Verhältnis von *partial-partial* zu *partial relations* beachten. Je nach Größe der oberen Schranke, die man für Reste R (vgl. Kapitel 4.3.3) wählt und der Größe der zu faktorisierenden Zahlen, haben sich Werte zwischen 3.5 und 5.5 für dieses Verhältnis als laufzeitoptimal erwiesen. Dies hängt aber auch von der Implementierung ab. Bei der 95-stelligen Zahl des folgenden Beispiels handelt es sich um einen Kofaktor von p(13693).

Beispiel für 95 Dezimalstellen

# fulls	# partials	LP	# partial-partials	DLP	DLP/ # fulls
520	7689	19	36096	20	0.04
1838	26478	321	122647	412	0.22
3031	51254	1613	210770	2111	0.70
4197	76079	3187	298374	4798	1.14
5345	105443	5561	392376	9672	1.81
6604	132591	7965	484053	16719	2.53
7361	151969	9927	546013	23316	3.16
8144	171365	11992	608180	31410	3.86
8956	190297	14175	668606	40889	4.57

Zykellänge

Länge	Anzahl	Länge	Anzahl	Länge	Anzahl
2	14175	9	846	16	23
3	7587	10	470	17	9
4	5912	11	295	18	6
5	4474	12	170	19	3
6	3192	13	94	20	2
7	2174	14	49	21	1
8	1376	15	29	22	2

Welche Auswirkungen bereits kleine Änderungen des Parameters T bei der Testwertbestimmung in der Auswertephase haben, zeigt der folgende Test. Es wurde für eine 100-stellige Zahl (Kofaktor von p(14591)) bei drei Siebintervallängen ($M_1 = 5 \cdot 10^6$, $M_2 = 6 \cdot 10^6$ und $M_3 = 7 \cdot 10^6$) einige charakteristische Daten für variierende Werte von T bestimmt. Es wurde jeweils die Anzahl der *partial* und *partial-partial relations* pro Zeit, deren Verhältnis, die Anzahl der durchschnittlich untersuchten Stellen in einem Teilsiebintervall der Länge 10^6 (Tests) und die Anzahl der untersuchten Stellen, die durchschnittlich benötigt wurden, um eine *partial-partial relation* zu finden. Dabei beziehen sich die Zeitangaben auf 1 Sekunde Rechenzeit auf einer Sparc ELC. Bei der Faktorisierung der Zahl wurden folgende Parameter gewählt :

$$M = 7 \cdot 10^6, \quad |FB| = 50\,000 \quad \text{und} \quad T = 3.7\ .$$

Auswirkungen des Variierens von T bei $M_1 = 5 \cdot 10^6$

T	$\frac{partials}{Sekunde}$	$\frac{partial-partials}{Sekunde}$	$\frac{partial-partials}{partials}$	TESTS	$\frac{\text{TESTS}}{partial-partials}$
3.40	$3.78 \cdot 10^{-3}$	$1.37 \cdot 10^{-2}$	3.60	0.63	18.71
3.50	$3.45 \cdot 10^{-3}$	$1.46 \cdot 10^{-2}$	4.23	1.18	23.11
3.60	$4.15 \cdot 10^{-3}$	$1.68 \cdot 10^{-2}$	4.04	1.11	26.08
3.70	$3.40 \cdot 10^{-3}$	$1.72 \cdot 10^{-2}$	5.05	1.66	37.00
3.80	$4.31 \cdot 10^{-3}$	$1.83 \cdot 10^{-2}$	4.25	2.16	43.85
3.90	$3.57 \cdot 10^{-3}$	$1.82 \cdot 10^{-2}$	5.11	3.20	62.04

Auswirkungen des Variierens von T bei $M_2 = 6 \cdot 10^6$

T	$\frac{partials}{Sekunde}$	$\frac{partial-partials}{Sekunde}$	$\frac{partial-partials}{partials}$	TESTS	$\frac{\text{TESTS}}{partial-partials}$
3.50	$4.04 \cdot 10^{-3}$	$1.34 \cdot 10^{-2}$	3.30	0.84	27.04
3.65	$4.23 \cdot 10^{-3}$	$1.69 \cdot 10^{-2}$	4.00	1.47	35.06
3.75	$3.26 \cdot 10^{-3}$	$1.55 \cdot 10^{-2}$	4.76	2.60	55.06
3.90	$3.01 \cdot 10^{-3}$	$1.64 \cdot 10^{-2}$	5.29	4.21	82.89
4.10	$2.19 \cdot 10^{-3}$	$1.51 \cdot 10^{-2}$	6.93	7.64	140.40

Auswirkungen des Variierens von T bei $M_3 = 7 \cdot 10^6$

T	$\frac{partials}{Sekunde}$	$\frac{partial-partials}{Sekunde}$	$\frac{partial-partials}{partials}$	TESTS	$\frac{\text{TESTS}}{partial-partials}$
3.50	$3.54 \cdot 10^{-3}$	$1.37 \cdot 10^{-2}$	3.89	0.66	19.61
3.60	$3.14 \cdot 10^{-3}$	$1,72 \cdot 10^{-2}$	5.47	1.59	34.52
3.70	$3.27 \cdot 10^{-3}$	$1.77 \cdot 10^{-2}$	5.45	2.29	46.72
3.90	$3.00 \cdot 10^{-3}$	$1.69 \cdot 10^{-2}$	5.66	4.27	82.05
4.10	$2.36 \cdot 10^{-3}$	$1.46 \cdot 10^{-2}$	6.23	7.61	169.43

Beispiel für 100 Dezimalstellen

# fulls	# partials	LP	# partial-partials	DLP	DLP/ # fulls
746	11633	208	63312	211	0.28
1716	25645	404	141339	482	0.28
3580	53414	1236	295052	2219	0.62
5166	76966	2436	418761	6232	1.21
6813	101533	4068	544544	15446	2.26
8233	123863	5927	648209	29242	3.55
9496	139629	7427	729085	43920	4.62

Zykellänge

Länge	Anzahl	Länge	Anzahl	Länge	Anzahl
2	7427	10	1134	18	19
3	7983	11	744	19	16
4	6997	12	491	20	6
5	6130	13	291	21	2
6	4678	14	190	22	4
7	3479	15	101	23	2
8	2478	16	64	24	0
9	1652	17	32	25	0

Als letztes Beispiel folgen die Daten, die bei der Faktorisierung einer 103-stelligen Zahl (Kofaktor von p(11177)) gesammelt wurden.

Beispiel für 103 Dezimalstellen

# fulls	# partials	LP	# partial-partials	DLP	DLP/ # fulls
216	9868	35	51580	41	0,19
451	20403	159	108386	197	0,44
729	32895	395	174960	548	0,75
1011	45388	727	241750	1139	1,13
1225	55194	1092	294012	1903	1,55
1562	69821	1737	373090	3717	2,38
1875	84288	2503	450785	6589	3,51
2104	94978	3148	508216	9692	4,61
2604	116804	4767	626364	20133	7,73
2936	130989	5988	702403	30369	10,34
3326	146866	7431	787024	45431	13,66
3610	158940	8686	850048	59196	16,40

Zykellänge

Länge	Anzahl	Länge	Anzahl	Länge	Anzahl
2	8686	10	1149	18	27
3	10242	11	748	19	10
4	8228	12	495	20	8
5	6577	13	292	21	5
6	4896	14	195	22	3
7	3586	15	113	23	1
8	2544	16	74	24	3
9	1716	17	35	25	0

Die folgenden Abbildungen stellen den typischen Verlauf einer Faktorisierung mit *Double Large Prime Variante (DLPV)* dar. Sie enthalten die gesamte Anzahl, der aus *partial* und *partial-partial relations* konstruierbaren *full relations*. Außerdem sind die direkt gefunden *full relations* fein gestrichelt eingezeichnet und die *full relations*, die durch die *Large Prime Variante* konstruiert werden konnten, sind grob gestrichelt eingezeichnet. In Abbildung 4.7 ist der Verlauf für eine 95-stellige Zahl (Kofaktor von p(13693)) eingezeichnet. Abbildung 4.8 gibt den Verlauf der Faktorisierung einer 103 stelligen Zahl (Kofaktor von p(11177)) wieder.

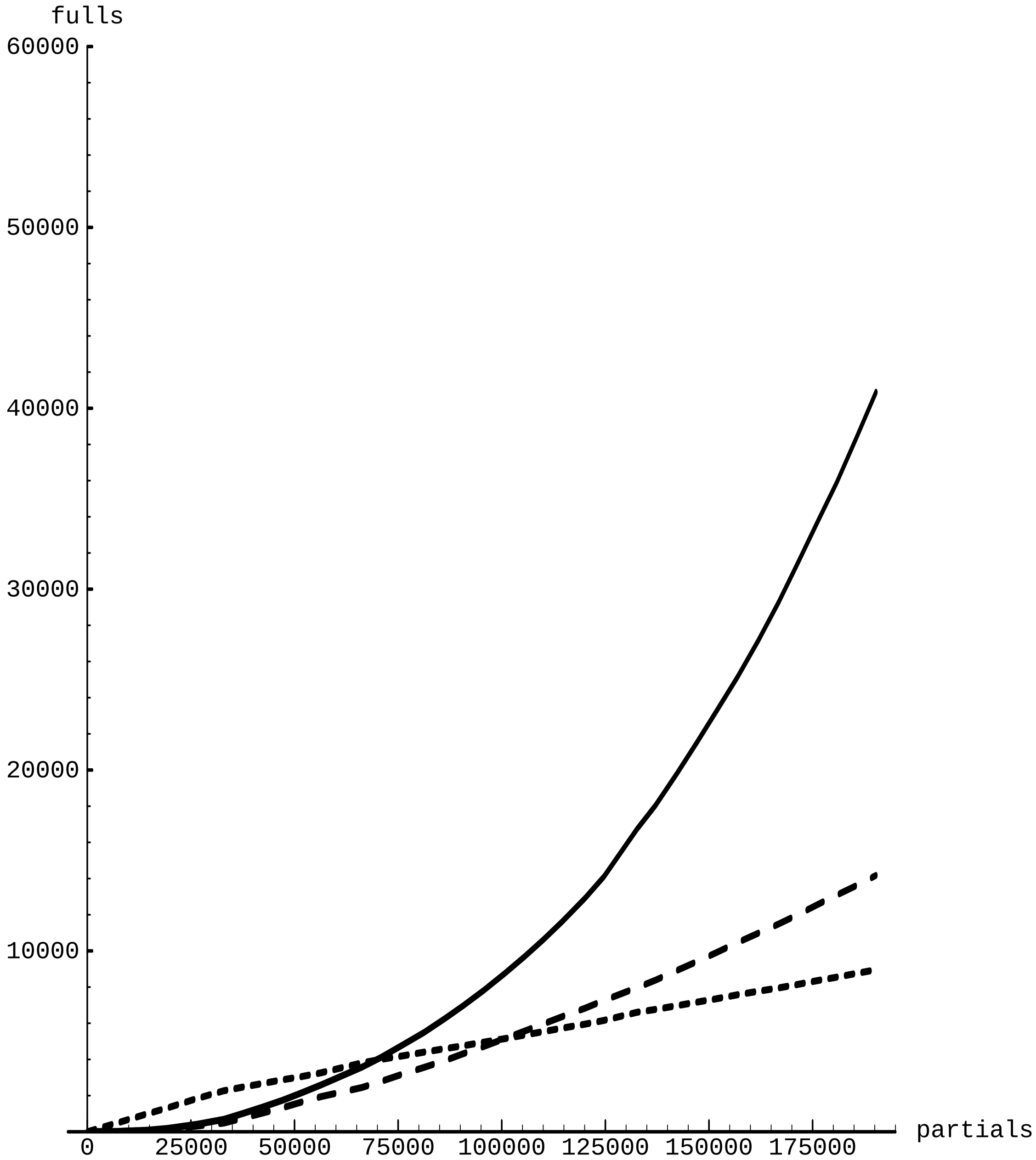


Abbildung 4.7: Verlauf der Faktorisierung mit *DLPV* bei 95 Stellen

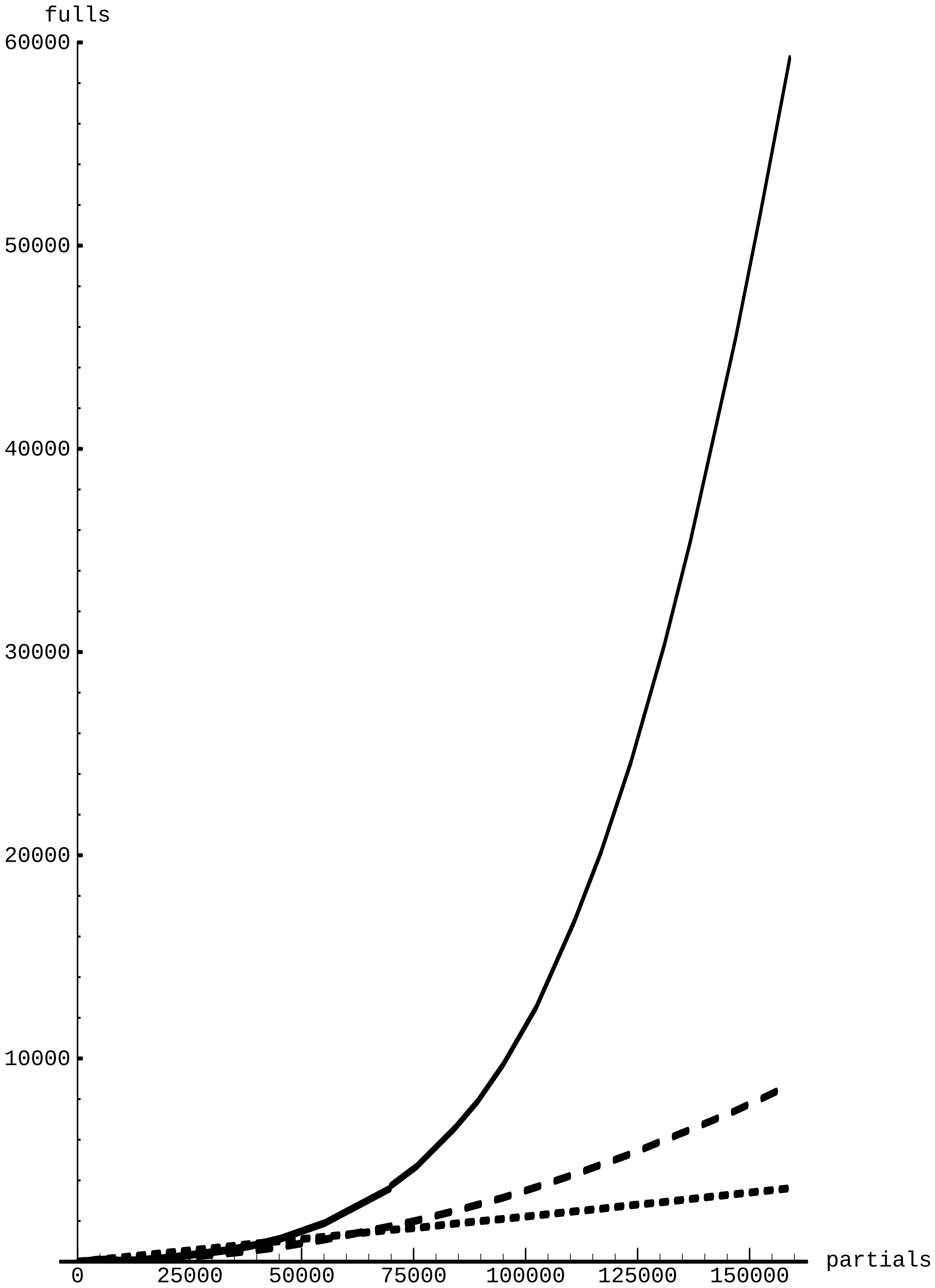


Abbildung 4.8: Verlauf der Faktorisierung mit *DLPV* bei 103 Stellen

Kapitel 5

Programmbeschreibung

In diesem Kapitel beschreibe ich die verschiedenen Versionen des Programms, deren Benutzung, die Formate der angelegten Dateien und einige Besonderheiten der Implementierung. Außerdem wird das Programm zur Verwaltung der gefundenen Relationen erläutert. Im Anhang A befinden sich die wichtigsten Module, die benötigten *makefiles* und die Headerdatei. In den verschiedenen Funktionen sind ihre Parameter und das prinzipielle Vorgehen erläutert. Die Funktionen, die nicht im Anhang A aufgeführt sind, werden kurz beschrieben.

Das Programm wurde in der Programmiersprache C für UNIX-Rechner entwickelt. Dabei wurde eine Langzahlarithmetik (Larith) benutzt. Diese Bibliothek sowie einige grundlegende mathematische Funktionen wurden von Stefan Düllmann ([Dül]) entwickelt. Diese beruhen haupsächlich auf den *Classical Algorithms* aus [Knu]. Zur Verteilung des Programms auf einem Netz von Sparc Workstations wurde die von Ralf Roth und Thomas Setz entwickelte LIPS Bibliothek (vgl. [LIPS]) verwendet.

Der Modulare Aufbau des Programms hält sich stark an die Einteilung des Programms in die verschiedenen Phasen (vgl. Kapitel 3.2). Es existieren 2 Versionen des Programms: eine verteilte Version für ein Netz von Sparc Workstations und eine single-Version für einen UNIX-Rechner. Die beiden Versionen des Programms benutzen im wesentlichen dieselben Module und Funktionen. Sie unterscheiden sich lediglich dort, wo dies durch das Verteilen des Algorithmus notwendig ist. Die single-Version dient hauptsächlich zum Austesten neuer Versionen der einzelnen Module. Beide Versionen benötigen als Eingabe lediglich die zu faktorisierende Zahl.

Um ein möglichst effizientes Programm zu erhalten, wird der Hauptspeicherbedarf des Programms direkt am Anfang allokiert und erst beim Programmende wieder freigegeben. Dadurch wird der Aufwand der Speicherverwaltung möglichst gering gehalten. Außerdem wird versucht, möglichst viel vorzuberechnen und alle Informationen, die zu Laufzeitverbesserungen führen könnten, zu berücksichtigen.

5.1 Verteilen des Programms

Das Quadratische Sieb eignet sich sehr gut zur Verteilung auf mehrere Rechner. Die verschiedenen Rechner und Teilprobleme sind unabhängig voneinander. Man berechnet auf jedem Rechner für verschiedene Polynome Relationen und sammelt diese auf einem speziellen Rechner. Man muß nur sicherstellen, daß kein Rechner ein Polynom benutzt, mit dem bereits ein anderer Rechner gearbeitet hat. Dies geschieht durch ein Master-Programm, das ein sternförmiges Netz von Clients aufbaut.

Auf dem Master werden die Vorberechnungen durchgeführt, die von allen Rechnern benötigt werden, und zu den Clients transportiert. Der Master verteilt die Teilaufgaben an die Clients, indem er Intervalle verschickt, aus denen jeder Client explizite Werte für D bestimmen kann. Dieser Parameter bestimmt eindeutig die Wahl der Polynome (vgl. Kapitel 3.4). Bei verschiedenen Intervallen ist somit sichergestellt, daß alle Clients paarweise verschiedene Polynome benutzen.

Am Anfang meiner Arbeit mit der verteilten Version berechnete der Master explizite Werte für D und verschickte eine bestimmte Anzahl an die Clients. Mit zunehmender Anzahl von Clients blockierte der Master jedoch das System, da er diese Werte nicht schnell genug berechnen konnte und die Clients darauf warten mußten. Deshalb verschickt der Master nun ein Intervall und jeder Client bestimmt sich sich selbst aus seinem Intervall Werte für D, die die an D gestellten Bedingungen erfüllen (vgl. (3.2)). Ist ein Intervall abgearbeitet, so teilt der Client dies dem Master mit (Message 1). Er übermittelt gleichzeitig einige statistische Daten. Der Master schickt ein neues Teilintervall und wertet die statistischen Informationen aus. Nach Erhalt des neuen Teilintervalls verschickt der Client die im letzten Intervall gesammelten Relationen und teilt dem Master danach mit Message 2 mit, daß die Übertragung abgeschlossen ist und der Master die Relationen jetzt übernehmen kann. Beide Messages sind entkoppelt, so daß der Master nicht durch einen Client, der Message 1 verschickt hat und dann ausfällt, bevor er Message 2 versenden konnte, blockiert wird. Nachdem der Master das Netz gestartet hat und jedem Client die vorberechneten Werte und ein Intervall für D verschickt hat, geht er in eine Schleife, in der er auf Messages der Clients wartet und diese bearbeitet. Der Master benötigt kaum Rechenzeit.

Wurden von den Clients eine bestimmte Anzahl Polynome verarbeitet, so veranlaßt der Master die Weiterverarbeitung der neu gesammelten Relationen. Er ruft das Programm *zusam* auf. Die Arbeitsweise dieses Programms wird später erläutert. Das Programm arbeitet unabhängig vom Master und ermittelt unter anderem auch die Anzahl der bisher direkt gefundenen und der konstruierbaren *full relations*. Ist diese Anzahl größer als die Anzahl der Elemente der Faktorbasis, so schließt der Master die Phase des Sammelns der Relationen ab. Er beendet die Programme auf den Clients und baut damit das Netz ab. Der Master konstruiert die *full relations*, die aus *partial relations* gebildet werden können, und erzeugt die Eingabe für das Programm zum Lösen des Gleichnungssystems. Nachdem das Gleichungssystem gelöst ist, bestimmt man mit dem Programm *wert_aus* aus den Lösungen die Teiler der Zahl N.

5.2 Programm zur Verwaltung der Relationen

Im Verlauf der Faktorisierung einer Zahl mit der *Double Large Prime Variante* werden einige hundert Megabyte an Daten gesammelt. Die im folgenden dargestellte Methode vereinfacht die Verwaltung dieser recht großen Dateien und reduziert Plattenbedarf bei der Übernahme neuer Relationen.

Der Master sammelt alle Relationen, die die Clients gefunden haben. Nachdem eine genügend größe Anzahl von Polynomen von ihnen bearbeitet wurde, ruft der Master das Programm *zusam* im Hintergrund auf. Die Anzahl der Polynome sollte nicht zu klein sein (das Zusammenschreiben sollte nicht unnötig oft geschehen), sie darf aber auch nicht zu groß werden (Beschränkungen beim Sortieren der Ergebnisdateien). Dieses Programm übernimmt dann die neu gesammelten Relationen und fügt diese in eventuell vorhandenen Relationen ein. Vorher wird jedoch getestet, ob die neuen Relationen korrekt sind. Danach sortiert man die neuen Relationen getrennt nach *full*, *partial* und *partial-partial relations*, wobei man eventuell doppelt vorkommende Relationen löscht.

Wurden bereits vor dem aktuellen Aufruf des Programms Relationen gesammelt, so stehen diese bereits sortiert und geordnet nach ihrem Typ in Dateien. *Full relations* werden in einer einzigen Datei und *partial relations* nach Typ geordnet in eventuell mehreren durchnumerierten Dateien, die höchstens 40000 Zeilen haben dürfen, gespeichert. Wurde mehr als eine Datei zur Abspeicherung für einen Typ benötigt, so existiert eine Referenzdatei, die die erste (und eventuell einzige) *large prime* der letzten Zeile aller vorhandenen Dateien enthält. Man sortiert nun die neuen Relationen in die bereits existierenden Dateien ein, so daß hintereinander betrachtet die Zeilen aller Dateien aufsteigend sortiert sind.

Nach diesem Schritt wird überprüft, ob eine Datei mehr als die zulässige Anzahl Zeilen enthalt. Ist dies der Fall, so wird diese Datei halbiert und die Referenzdatei aktualisiert. Dabei wird sichergestellt, daß alle Relationen eines Typs mit den gleichen ersten *large primes* in einer Datei stehen und nicht auf zwei Dateien aufgeteilt werden.

Hat man alle neuen Relationen in die bestehenden Dateien eingefügt, ermittelt man die Anzahl der kombinierbaren *full relations* mit dem Algorithmus aus Kapitel 4.3.1. Das Ergebnis wird in eine Datei geschrieben. Es bestehend aus der Anzahl der direkt gefunden *full relations*, der Anzahl der *partial relations*, der Anzahl der daraus konstruierbaren *full relations*, der Anzahl der *partial-pasrtial relations* und der Anzahl der insgesamt durch die *Double Large Prime Variante* konstruierbaren *full relations*. Diese Datei wird vom Master vor jeder Kommunikation gelesen und ausgewertet, so daß ihm bei der nächsten Kommunikation nach Beendigung des Programms *zusam* die aktualisierte Anzahl bekannt ist.

5.3 Benutzte Dateien und deren Format

FAKTORBASIS

Diese Datei enthält die Faktorbasiselemente und einige Daten der Faktorbasis. Dazu gehört in der ersten Zeile die Größe der Faktorbasis und in der zweiten Zeile die Zahl, für die diese Faktorbasis erstellt wurde. In dieser Zahl ist der jeweils gewählte Multiplier enthalten. Dann folgen die einzelnen Faktorbasiselemente jeweils in einer Zeile. In der letzten Zeile der Datei steht als Abschluß eine 0.
Beispiel:

```
55000
14528617204966389476449999...
-1
2
3
5
7
11
⋮
1445953
1445981
1446019
1446023
0
```

FAKTORENMATRIX

In dieser Datei werden während der Phase des Sammeln der Relationen die direkt gefundenen *full relations* abgespeichert. In der verteilten Version hängt jeder Client seinen Namen an diese Datei an.
(Beispiel: FAKTORENMATRIX.crypt14.cs.uni-sb.de)

Dadurch werden Zugriffsprobleme im Filesystem vermieden.
Für eine *full relation*

$$X^2 \equiv \prod_{p_i \in FB} p_i^{e_i} \bmod N$$

wird X und die Folge von Paaren $(e_j\ ,\ j)$ mit $e_j \neq 0$ für $j = 1, \dots, |FB|$ abgespeichert. Dabei wird der Wert von X und die Folge durch einen Doppelpunkt getrennt. Am Ende der Folge steht eine 0. Zur Abspeicherung von Zerlegungen über der Faktorbasis wird generell dieses Schema verwendet.

Beispiel :
10606355115609535477672825034379844071373754186247287553622497771627892804136716654819501719580931568270185666985148678 29 : 1 1 1 2 1 3 1 9 1 25 1 48 1 78 1 245 1 252 1 628 1 963 1 1935 1 4940 1 17155 1 62140 1 70432 1 88412 1 170262 0

PARTION

In dieser Datei werden die *partial relations* gespeichert. Analog zur Datei **FAKTORENMATRIX** hängen die Clients ihren Namen an. Für eine *partial relation*

$$X^2 \equiv q \cdot \prod_{p_i \in FB} p_i^{e_i} \bmod N$$

wird q, X und die Zerlegung über der Faktorbasis wie oben abgespeichert. Als Trennzeichen zwischen der *large prime* q und X dient das Symbol @.
Beispiel :
9947299 @ 1482504018065051412290373564229610401065493851427750764738
928846908479600891326060769831792003931633970040696397616808083 : 1 1
4 2 1 147 1 266 1 356 1 372 1 1011 1 3811 1 5552 1 13488 1 18017 1
70407 1 84042 1 95215 0

PART.LARGE

In dieser Datei werden die *partial-partial relations* gespeichert. Analog zur Datei **FAKTORENMATRIX** hängen die Clients ihren Namen an. Für eine *partial-partial relation*

$$X^2 \equiv q_1 \cdot q_2 \cdot \prod_{p_i \in FB} p_i^{e_i} \bmod N$$

wird q_1, q_2, X und die Zerlegung über der Faktorbasis wie oben abgespeichert. Als Trennzeichen zwischen den *large primes* und X dient @. Die beiden *large primes* werden durch ein Leerzeichen getrennt.
Beispiel :
7432987 162320003 @ -100106727660857830362759959757264202674530716680
9128945742259050351851871733140468008431453656411948455209580495370
400104 : 2 2 1 3 1 20 1 1550 1 5726 1 23723 1 53908 1 78256 1 98921 1
124171 1 240089 0

LAST_D

Diese Datei enthält die obere Grenze des letzten vom Master verschickten Intervalls, aus dem ein Client Werte für D wählen kann. Sie wird vom Master nach Verschicken eines neuen Intervalls aktualisiert. Damit kann sichergestellt werden, das die Clients D immer aus verschiedenen Intervallen wählen und dadurch auch verschiedene Polynome benutzen.

PARAMETER

Eine zentrale Bedeutung kommt der Datei *PARAMETER* zu. Sie enthält in Abhängigkeit von der Anzahl der Dezimalstellen der zu faktorisierenden Zahl die zu wählenden Parameter. Diese sind der "Toleranzwert" T für die Bestimmung des Testwertes (vgl. Kapitel 3.6), die Siebgrenze M und die Anzahl der Faktorbasiselemente $\#FB$. Am Anfang jeder Zeile steht die Anzahl der Dezimalstellen der zu faktorisierenden Zahl, gefolgt von den dazugehörigen Parametern. Diese sind jeweils mit einem Komma von dem vorherigen Wert getrennt. Zeilen, an deren Anfang ein "#" steht, werden überlesen.

```
# Parameter fuer PPMPQS
# Die Datei ist wie folgt aufgebaut:
# Stellenzahl, Toleranzwert, Siebgrenze, # Faktorbasiselemente
          66, 1.7, 1000000, 5300
          67, 1.9, 1000000, 5800
          68, 1.9, 1000000, 6100
          69, 2.0, 1000000, 6800
          70, 2.0, 1000000, 7400
          71, 2.0, 1000000, 8000
          72, 2.0, 1000000, 8500
          73, 2.0, 1000000, 9000
          74, 2.0, 1000000, 9500
          75, 2.5, 1500000, 10200
          76, 2.9, 1500000, 11000
          77, 3.0, 2000000, 12000
          78, 3.0, 2000000, 13000
          79, 3.0, 2000000, 14000
          80, 3.1, 3000000, 15000
          81, 3.1, 3000000, 16000
          82, 3.1, 3000000, 17000
          83, 3.1, 3000000, 18500
          84, 3.2, 3000000, 20000
          85, 3.2, 3000000, 21500
          86, 3.2, 3000000, 23000
          87, 3.2, 3000000, 25000
          88, 3.3, 3000000, 27000
          89, 3.3, 3000000, 28000
          90, 3.5, 4000000, 30500
          91, 3.6, 4000000, 32000
          92, 3.6, 4000000, 35000
          93, 3.7, 4000000, 37000
          94, 3.7, 4000000, 39500
          95, 3.7, 5000000, 41500
          96, 3.8, 5000000, 45000
          97, 3.8, 5000000, 47500
          98, 3.7, 6000000, 51000
          99, 3.8, 6000000, 53000
          100,3.8, 7000000, 51000
          101,3.8, 7000000, 54000
          102,3.8, 7000000, 57000
          103,3.9, 8000000, 61000
          104,3.9, 8000000, 66000
          105,3.9, 8000000, 70000
          106,3.9, 8000000, 75000
          107,3.9, 8000000, 80000
```

RESSOLW

Diese Datei enthält die Quadratwurzeln der zu faktorisierenden Zahl modulo aller Faktorbasiselemente. Die Quadratwurzeln wurden mit dem von D. Shanks entwickelten RESSOL-Algorithmus berechnet.

CLCONF

Diese Datei enthält Informationen über die benutzten Clients. Sie wird beim Start der verteilten Version des Programms vom Lips-System (vgl.[LIPS]) benötigt.

RELATIONS

Diese Datei dient als Eingabe für das Gauss-Eliminationsprogramm zum Lösen des Gleichungssystems (2.5). Ihr Format ähnelt dem der Datei **FAKTORENMATRIX**. Am Anfang steht eine fortlaufende Nummer (beginnend mit 1) und die doppelte Anzahl der Paare der Folge, die zur Abspeicherung der Zerlegung über der Faktorbasis verwendet wird. Begrenzt von je einer 0, folgt diese dann. Stünde die *full relation* aus dem Beispiel der **FAKTORENMATRIX** in der 1. Zeile, so sähe ihr Eintrag in der Datei **RELATIONS** folgendermaßen aus :
1 36 0 1 1 1 2 1 3 1 9 1 25 1 48 1 78 1 245 1 252 1 628 1 963 1 1935 1 4940
1 17155 1 62140 1 70432 1 88412 1 170262 0

5.4 Funktionsbeschreibung

Hier folgt eine kurze Beschreibung der Funktionalität der benutzten Funktionen, die nicht in Anhang A aufgelistet sind oder aus einer der beiden verwendeten Bibliotheken stammen. Dabei bezeichnen Großbuchstaben Variablen für "lange" Zahlen. Variablen mit einem elementaren Datentyp werden mit kleinen Buchstaben bezeichnet.

int linvers(A, B, ERG)

Die Funktion berechnet das Inverse einer "langen" Zahl A modulo einer "langen" Zahl B. Das Ergebnis wird in der "langen" Zahl ERG zurückgegeben. Die Funktion liefert:

$$\begin{array}{rl} 0, & \text{falls die Berechnung erfolgreich war} \\ -1, & \text{falls kein Inverses existiert} \end{array}$$

int sinvers(a, b)

Die Funktion berechnet das Inverse einer "kurzen" Zahl a modulo einer "kurzen" Zahl b. Das Ergebnis wird als kurze Zahl zurückgegeben.
Die Funktion liefert:

$$\begin{array}{rl} a \bmod b, & \text{falls die Berechnung erfolgreich war} \\ -1, & \text{falls kein Inverses existiert} \end{array}$$

int m_ressol(a, p)

Die Funktion berechnet eine Quadratwurzel von a modulo einer Primzahl p mit dem RESSOL-Algorithmus von D. Shanks. Das Ergebnis wird zurückgegeben.

float mips()

Die Funktion führt den Befehl *hostid* aus und versucht, aus den beiden ersten Ziffern des Ergebnises die MIPS-Leistung der Maschine zu bestimmen.
Die Funktion liefert:

MIPS-Leistung,	falls Bestimmung möglich war
1,	sonst

int get_parameter(stellen, t, m, groesse_FB)

Die Funktion liest aus der Datei **PARAMETER** die zu einer Dezimalstellenzahl (stellen) gehörenden Parameter ein.
Die Funktion liefert:

0,	falls die Parameter korrekt gelesen wurden
-1,	sonst

int legendre(N, p)

Die Funktion berechnet das Legendre-Symbol einer "langen" Zahl N und einer Primzahl p.
Die Funktion liefert:

1,	falls N quadratischer Rest modulo p ist
0,	falls $\mathrm{ggT}(N, p) > 1$
-1,	falls N quadratischer Nichtrest modulo p ist

int fermat(N)

Die Funktion führt den Primzahltest nach Fermat für eine "lange" Zahl N durch. Dabei werden die Basen 2, 3, 5 und 7 benutzt.
Die Funktion liefert:

1,	falls N Pseudoprimzahl ist
0,	falls N als zusammengesetzt erkannt wurde

int listprim(N, anzahl)

Die Funktion führt den Primzahltest nach Miller-Rabin für eine "lange" Zahl N durch. Dabei gibt werden *anzahl* viele Basen benutzt.
Die Funktion liefert:

1,	falls N Pseudoprimzahl ist
0,	falls N als zusammengesetzt erkannt wurde

int squfof(N,LP, p1 ,p2)

Die Funktion versucht, die beiden Teiler der "langen" Zahl N mit SQUFOF zu berechnen. Dabei werden nur Zerlegungen akzeptiert, bei denen beide Teiler $p1, p2$ kleiner als LP sind.
Die Funktion liefert:

2,	falls die gewünschte Zerlegung gefunden wurde
0,	falls einer der Teiler zu groß ist
1,	sonst

int pollard(N, LP, p1 ,p2)

Die Funktion versucht, die beiden Teiler der "langen" Zahl N mit Pollards ρ-Methode zu berechnen. Dabei werden nur Zerlegungen akzeptiert, bei denen beide Teiler $p1, p2$ kleiner als LP sind.
Die Funktion liefert:

2,	falls die gewünschte Zerlegung gefunden wurde
0,	sonst

5.5 Laufzeitverteilung und Verbesserungen

Aus einem kurzen Ausschnitt einer Faktorisierung, der mit der single-Version des Programms erstellt wurde, erhält man die Verteilung der Laufzeit auf die einzelnen Module.

Name	#Aufrufe	user	system	user+sys	Gesamtanteil
bestimme_FB	1	2448	36	2484	0.19 %
berechne_ko.	494	322972	1250	324222	25.07 %
sieben	494	826860	2257	829117	64.11 %
teste	494	136510	883	137393	10.62 %
gesamt:		1288790	4426	1293216 = 3 h 35 min 32 sec	

Das Beispiel wurde für eine 100-stellige Zahl mit Siebgrenze $M = 5 \cdot 10^6$, $T = 3.7$ und $\#FB = 50\,000$ berechnet. Es wurden insgesamt 494 Polynome untersucht. Die Spalten `user`, `system` und `user+sys` enthalten die benötigte User-Zeit, Systemzeit und deren Summe. Man erkennt sofort die Dominanz der Siebphase. Anhand dieser Verteilung kann man auch ermitteln welchen Einfluß die Verbesserung eines Teilmoduls auf die gesamte Laufzeit hat.

Eine mögliche Verbesserung des Algorithmus stellt ein zweistufiges Sieb dar. In einem ersten Schritt wird nur mit großen Faktorbasiselementen (z.B. p_i mit $i > 5000$)

gesiebt. Dann bestimmt man die Siebeinträge, die einen Testwert übersteigen. In der zweiten Phase siebt man wie normal (mit allen Faktorbasiselementen) bis zu den Stellen mit einem Siebeintrag größer als der ersten Testwert. Nun kann man die Teiler des zugehörigen Funktionswertes direkt durch Vergleiche bestimmen und spart sich die ursprüngliche Auswertung der Siebphase. Die Effektivität dieser Methode hängt davon ab, wie stark die Zerlegungen durch große Faktorbasiselemente bestimmt werden und wie stark der ersten Testwert abgesenkt werden muß, damit man beim zweiten Sieben keine Stellen vergißt, die zu einer Relation führen würden.

Der Wahl der Polynome kommt eine entscheidende Rolle beim Algorithmus zu. Sie bestimmt die Größe der zu faktorisierenden Funktionswerte. Es existieren noch Alternativen zu der in Kapitel 3.4 dargestellten Möglichkeit (z.B. [Pom, S.35]). Es sollte außerdem überprüft werden, ob ein anderer Ansatz als das Anpassen der Polynome an ein fast symmetrisches Intervall die erhöhte Wahrscheinlichkeit einer Zerlegung um die Nullstellen besser ausnutzen kann.

Im Moment werden alle Zahlen, die zur Abspeicherung von Relationen benutzt werden, als ASCII-Zeichen abgespeichert. Wegen des hohen Platzbedarfs sollte eine kompaktere Art der Darstellung gewählt werden. Dies ginge zwar zu Kosten der direkten Lesbarkeit der Relationen, kann aber zu Reduzierung des Platzbedarfs von ungefähr 50 % führen. Eine Lösung wäre, die Abspeicherung der binären Darstellung.

Kapitel 6

Praktische Erfolge

Die Laufzeit des Quadratischen Siebes hängt (im Gegensatz zu der von ECM (vgl. [Ber]) nicht von der Größe der Teiler, sondern von der Größe der zu faktorisierenden Zahl ab. Das Quadratische Sieb ist im Moment der schnellste Faktorisierungsalgorithmus für zusammengesetzte Zahlen allgemeiner Form zwischen 90 und 120 Dezimalstellen, die keine kleinen Faktoren (kleiner als 40 Dezimalstellen) haben. Bei der Faktorisierung größerer Zahlen stellt das von mir entwickelte Programm außerdem eine sehr gute Ergänzung zu dem von Franz-Dieter Berger implementierten ECM-Programm dar. ECM findet sehr schnell kleinere Faktoren in den zu faktorisierenden Zahlen. Gelingt es ECM nicht, alle Faktoren einer Zahl zu ermitteln und ist der unfaktorisierte Rest in einem praktikablen Bereich für das Quadratische Sieb, so gelingt es einer Kombination beider Programme, die Zahl komplett zu faktorisieren. Beiden Programmen allein wäre dies nicht möglich gewesen. Diese Zusammenarbeit verläuft recht erfolgreich. Dies belegen unsere Erfolge beim “RSA Factoring Challenge”, einem internationalen Wettbewerb der Firma “RSA Data Security”, dessen Ziel es ist die weltweiten Fortschritte beim Faktorisieren großer Zahlen festzuhalten und die Forschung auf diesem Gebiet voranzutreiben.

6.1 Der RSA Faktorisierungs-Wettbewerb

Seit dem 1.4.1991 wird dieser Wettbewerb veranstaltet. Er basiert auf zwei Zahlenlisten - der RSA-Liste und der Partitions-Liste. Beide Listen enthalten Zahlen ab 100 Dezimalstellen. Die folgenden Informationen entstammen einem ausführlicheren Informationsschreiben, daß man per Email vom Challenge Administrator anfordern kann. (Mail an *challenge-administrator@rsa.com*)

Die RSA-Liste enthält Zahlen der Art, wie sie in RSA-Cryptosystemen verwendet werden. Diese Zahlen bestehen aus zwei ungefähr gleichgroßen Primfaktoren. Man vermutet, daß diese Zahlen am schwierigsten zu faktorisierenden sind. Die größte bisher (Dezember 1992) faktorisierte RSA-Zahl hat 110 Dezimalstellen und wurde am 14.4.1992 von Arjen K. Lenstra faktorisiert. Im Moment arbeiten Arjen K. Lenstra, Bruce Dodson, Mark Manasse und ich an der Faktorisierung der nächsten RSA-Zahl in der Liste. Diese Zahl hat 120 Dezimalstellen und ihre Faktorisierung würde einen neuen weltweiten Faktorisierungsrekord mit dem Quadratischen Sieb darstellen. Wir hoffen, die Zahl Mitte diesen Jahres faktorisiert zu haben.

Die Partitions-Liste enthält Partitionszahlen $p(n)$ (vgl. Definition 1.10). Betrachtet man die Schwierigkeit, diese Zahlen zu faktorisieren, so scheinen sie sich mehr oder weniger zufällig zu verhalten. Das bedeutet, daß es einige Zahlen in dieser Liste gibt, die relativ einfach zu faktorisieren sind und andere Zahlen, deren Faktorisierung sich als äußerst schwierig erweist. Die kleinste Partitionszahl in der Liste ist $p(8681)$. Die Liste ist nach oben offen und es wurden nur Partitionszahlen $p(x)$ aufgenommen, deren Argumente Primzahlen $x \geq 8681$ sind.

Am Ende jedes Quartals wird das ausgesetzte Preisgeld von 1750\$ unter den erfolgreichen Teilnehmern aufgeteilt. Dazu wird folgendes Punktesystem verwendet: Für jede Faktorisierung einer bis dahin nicht faktorisierten Partitionzahl erhält man Punkte. Die Anzahl der Punkte hängt von der Differenz der Dezimalstellen der Zahl und der kleinsten noch nicht faktorisierten Zahl ab.
Sei D die Anzahl der Dezimalstellen, der kleinsten noch nicht faktorisierten Partitionszahl. Für die Faktorisierung einer Zahl mit $D + d$ Dezimalstellen erhält man

$$\begin{array}{llrcccr} 8 \text{ Punkte}, & \text{falls} & 0 & \leq & d & \leq & 9, \\ 4 \text{ Punkte}, & \text{falls} & 10 & \leq & d & \leq & 19, \\ 2 \text{ Punkte}, & \text{falls} & 20 & \leq & d & \leq & 29, \\ 1 \text{ Punkt}, & \text{falls} & 30 & \leq & d & \leq & 49. \end{array}$$

Die Höhe des Gewinns am Ende des Quartals berechnet sich folgendermaßen:

$$\text{Gewinn} = 1750\$ \cdot \frac{\text{Anzahl von eigenen Punkte in diesem Quartal}}{\text{Gesamtanzahl aller Punkte in diesem Quartal}}$$

Es folgen unsere bisherigen Erfolge in diesem Wettbewerb, wobei das Quadratische Sieb erst im 2. Quartal 1992 eingesetzt werden konnte.

```
Partition Challenge List:
-------------------------
        (1st Quarter, 1992)
                122 points, Arjen Lenstra, $1405
                 12 points, Franz-Dieter Berger, Johannes Buchmann, $138
                  8 points, Peter L. Montgomery, $92
                  6 points, Peter L Montgomery & Paul C Leyland, $69
                  4 points, Lambert E. Bright, $46
                --------------
                152 points total
```

```
(2nd Quarter, 1992)
         78 points, Arjen Lenstra, $989
         48 points, Franz-Dieter Berger, Thomas Denny,
                    Johannes Buchmann, $609
          5 points, Peter L. Montgomery, $63
          4 points, Peter L Montgomery & Paul C Leyland, $51
          3 points, Paul C Leyland, $38
        --------------
        138 points total

(3rd Quarter, 1992)
          1 point, Heinz-Bernd Eggenstein, $11
         84 points, Arjen K. Lenstra, $907
          1 point, Paul C Leyland & Arjen K Lenstra, $11
         58 points, Franz-Dieter Berger, Thomas Denny,
                   Johannes Buchmann, $626
         12 points, Paul C Leyland, $130
          2 points, Francois Morain, Reynald Lercier, $22
          4 points, FactOregon (Joe Buhler, Peter Montgomery,
                               Robby Robson, Russell Ruby), $43
        --------------
        162 points total

(4th Quarter, 1992)
         79 points, Arjen K. Lenstra, $634
          4 points, Paul Leyland, Arjen K. Lenstra, $32
          4 points, Bruce Dodson, Arjen K. Lenstra, $32
         60 points, Franz-Dieter Berger, Thomas Denny,
                    Johannes Buchmann, $482
          6 points, FactOregon (Joe Buhler, Peter Montgomery,
                                Robby Robson, Russell Ruby), $48
         52 points, Bruce Dodson, Matt Fante, $418
          5 points, Paul C Leyland, $40
          3 points, Heinz-Bernd Eggenstein, $24
          5 points, Reynald Lercier, Francois Morain, $40
        --------------
        218 points total
```

Ein gutes Beispiel für die erfolgreiche Zuzsammenarbeit von ECM und QS liefert die Partitionszahl $p(13723)$:

$$
\begin{aligned}
\mathrm{p}(13723) = \;& 33193986308108491604447811773637423416031181706611478556238148708372443 \\
& 25092885333165199191538446220320718612375562494423963\ (126) \\
= \;& 11 * 11 * 11 && \text{(Probedivision)} \\
* \;& 16915499\ (9) && \text{(ECM)} \\
* \;& 4041033318662479\ (16) && \text{(ECM)} \\
* \;& 191809443001459398605506549864988727083612717791\ (48) && \text{(QS)} \\
* \;& 19021034171871381981680816165988526856566571184715643\ (53) && \text{(REST)}
\end{aligned}
$$

Ohne die Vorarbeit von ECM wäre es mit dem QS nicht möglich gewesen, diese Zahl zu faktorisieren. Auf der anderen Seite ist es ECM auch nicht möglich, einen 48-stelligen Faktor zu finden.

Von den 60 Partitionszahlen, die von uns bisher bis zum 2. Quartal 1993 faktorisiert wurden, konnten 29 allein mit ECM faktorisiert werden. Die meisten Zahlen, die mit dem QS faktorisiert wurden, hatten zwischen 95 und 104 Dezimalstellen. Der größte Kofaktor einer Partionszahl, der von mir mit dem QS-Programm faktorisiert wurde, hatte 104 Dezimalstellen. Die Faktorisierung war am 9.2.93 beendet.

p(12497)	=	394325053319473486830567441649120997667810384566754128785701231621420452335446937917470518428762296737965208663000187627 (120)	
	=	20599	(Probedivision)
	∗	769266920329 (12)	(ECM)
	∗	266645834863245909562783789005048009581 (39)	(QS)
	∗	93324650340132260139277695225473850712127365716671026543061024377 (65)	(REST)

Die dafür benötigte Zeit betrug 50 MIPS-Jahre. Einen Einblick in die Möglichkeiten der Optimierung des Algorithmus durch steigende praktische Erfahrungen mit der *Double Large Prime Variante* erhält man, wenn man diese Faktorisierung mit der Faktorisierung von $p(13751)$ vergleicht. Diese Faktorisierung wurde im Oktober 1992 durchgeführt und dort benötigte der Algorithmus zur Faktorisierung einer 100-stelligen Zahl 60 Mips-Jahre.

p(13751)	=	450031558360750694116941385295075484304570908103270380843155898277503469963878986690851166333276018911484193336765945931453338 (126)	
	=	2 ∗ 3 ∗ 103 ∗ 601 ∗ 13613	(Probedivision)
	∗	29339731382533843 (17)	(ECM)
	∗	8986849103974067059940017956689 (31)	(QS)
	∗	337569029422749531307844715292827582671106452764416333540329488227991 (69)	(REST)

Bedenkt man, daß eine Erhöhung der Dezimalstellen der zu faktorisierenden Zahl eine Verdoppelung der Laufzeit nach sich zieht, so konnte die Laufzeit des Algorithmus in dieser Zeit um einen Faktor von 2.4 verkürzt werden.

Um den aktuellen Stand des Wettbewerbs zu erhalten, schickt man eine Email an *challenge-news@rsa.com*. Dabei kann der Inhalt der Mail leer sein. Es wird lediglich der Absender der Nachricht ermittelt und an dessen Adresse eine Datei mit dem aktuellen Stand gesendet.

Anhang A

Listings

A.1 Hauptprogramme

A.1.1 Der Master

Das Makefile

```
################################################################################
#                                                                              #
#                             Makefile fuer PMPQS                              #
#                                                                              #
################################################################################

VER=4
MACHINE=-sun$(VER)

INCLUDE=/home/denny/INCLUDE
LIBDIR=/home/denny/lib
LIBS=$(LIBDIR)/sieve.a /home/berger/DIPLOM/LARITH/larith.a
     /home/thsetz/lips_2.2F/lib/liblips.a

SLIBS= -lm

OBJ=  pmpqs.o faktor_N.o gausvor.o zykelfind.o auswahl.o

OPT= -O4
.KEEP_STATE:

tlpmpqs: $(LIBS) $(OBJ)
cc $(OPT) $(MACHINE) -o tlpmpqs $(OBJ) $(LIBS) $(SLIBS)

pmpqs.o: pmpqs.c
cc $(OPT) $(MACHINE) -I$(INCLUDE) -c pmpqs.c

faktor_N.o : faktor_N.c
cc $(OPT) $(MACHINE) -I$(INCLUDE) -c faktor_N.c
```

```
auswahl.o : auswahl.c
cc $(OPT) $(MACHINE) -I$(INCLUDE) -c auswahl.c

zykelfind.o : zykelfind.c
cc $(OPT) $(MACHINE) -I$(INCLUDE) -c zykelfind.c

gausvor.o : gausvor.c
cc $(OPT) $(MACHINE) -I$(INCLUDE) -c gausvor.c
```

Das Headerfile hmpqs.h

```
/*****************************************************************************/
/* DOC                         hmpqs.h                                       */
/*                                                                           */
/* Headerfile mit internen Deklarationen                                     */
/*                                                                           */
/*****************************************************************************/

#include <stdio.h>
#include <math.h>
#include <string.h>
#include <values.h>
#include "/home/berger/DIPLOM/LARITH/larith1.h"
#include "/home/thsetz/lips_2.2F/include/lips.h"

#define MAXSTELLEN      100
#define MAXSTRLEN       200
#define LOGMUL          1.3
#define BIS_PRIMZAHL    100
#define CONFIGFILE      "config"
#define DEFAULT_LENGTH  40
#define LARGEP          100000000
#define DOUBLEP         77300000
#define kleinstart      22
#define WURZINT         46300
#define SIEBINTERN      1000000

typedef unsigned char   SIEBTYP;
typedef unsigned short  EXPOTYP;
typedef unsigned long   *LZ;

double  ldouble();
double  log2();

#ifndef NULL
#define NULL            0
#endif

#ifndef TRUE
#define TRUE        1
#define FALSE       0
#define ERROR       -1
#endif

#ifndef max
#define max(a,b)    ((a) > (b) ? (a) : (b))
#define min(a,b)    ((a) < (b) ? (a) : (b))
#endif
```

pmpqs.c

```
#include "hmpqs.h"

/****************************************************************************/
/* DOC                                                                      */
/*      H A U P T P R O G R A M M    PPMPQS                                 */
/*                                                                          */
/*  zuletzt geaendert am 15.03.93                                           */
/****************************************************************************/
main(argc, argv)
     int argc;
     char *argv[];
{
  LONGTYPE N , teiler1, teiler2, kwurz_N;
  int stell, stellenzahl, rcode, k;
  char hlp[MAXSTELLEN];

/****************************************************************************/
/*  Initialisieren der lange Arithmetik                                     */
/****************************************************************************/

  linit(DEFAULT_LENGTH);

/****************************************************************************/
/* Einlesen der zufaktorisierenden Zahl N und Ermittelung des Speicherbedarfs */
/* in 4 Bytes Bloecken zur Abspeicherung der Zahl N. Dabei wird folgende     */
/* Formel, verwandt : log10(2) * (Dezimallaenge der Zahl) / 32              */
/* Zur Sicherheit addiere ich noch 4 Bloecke mehr als berechnet wurden .     */
/****************************************************************************/

  if (argc == 2)
    {
      stellenzahl = strlen(argv[1]);
      stell = (int)(2.302585*stellenzahl/32) + 4;
      N = lalloc(stell);
      string_to_l(argv[1], N);
    }
  else
    {
      while (1)
        {
          printf("Zu faktorisierende Zahl : ");
          gets(hlp);
          stellenzahl = strlen(hlp);
          if (strspn(hlp,"0123456789") == stellenzahl)
            break;
         }
      stell = (int)(2.302585*stellenzahl/32) + 4;
      N = lalloc(stell);
      string_to_l(hlp,N);
    }
  if (stellenzahl < 0)
    fehler(stellenzahl);
  lwrite(N);
  printf("Stellenzahl : = %d\n",stel(N));

/****************************************************************************/
/*   Pruefe ob die Zahl N eine Primzahl ist  ( mit Fermattest )             */
/****************************************************************************/

  if (fermat(N))
    printf("Die Zahl ist Primzahl\n");
  else
    {
```

```
/*****************************************************************************/
/*   Speicherallokieren fuer Teiler und Wurzel von N                         */
/*****************************************************************************/

      kwurz_N = lalloc(stel(N) +2);
      teiler1 = lalloc(stel(N) +2);
      teiler2 = lalloc(stel(N) +2);

/*****************************************************************************/
/*   Test ob N Quadratzahl ist                                               */
/*****************************************************************************/

      if (!quad_test (N , kwurz_N))
        {

/*****************************************************************************/
/*   Aufruf der Funktion faktorisiere_N nachdem getestet wurde, ob die Zahl   */
/*   mit dem QS faktorisiert werden kann.                                    */
/*   Die Funktion liefert 2 Teiler zurueck.
/*****************************************************************************/

          if (( rcode = faktorisiere_N(N, teiler1, teiler2)) == 0)
            {
              printf("Teiler gefunden :\n");
              lwrite(teiler1);
              lwrite(teiler2);
            }
          else
            printf("Kein Teiler gefunden\n");
         }
        else
          {
            llsto(kwurz_N, teiler1);
            printf("Die Zahl N ist ein Quadrat\n");
            printf("Teiler : ");
            lwrite(teiler1);
          }
    }
  free(N);
  free(kwurz_N);
  free(teiler1);
  free(teiler2);
}
```

faktor_N.c

```
#include "hmpqs.h"

/*****************************************************************************/
/* DOC                       FAKTORISIERE_N()                                 */
/*                                                                           */
/*      Die Funktion versucht die Zahl N mit dem QS-Sieb-Algorithmus zu       */
/*      faktorisieren. Sie ruft die dazu notwendigen Module auf.              */
/*                                                                           */
/*      Eingabe:  LONGTYPE N           = zu fakt. Zahl                        */
/*                                                                           */
/*      Ausgabe:  LONGTYPE *teiler?    = Teiler der Zahl N                    */
/*                                                                           */
/*      Return :  0                    = ohne Fehler                          */
/*                fehlernr             = sonst                                */
/*                                                                           */
/*      zuletzt geaendert am 06.07.92                                         */
/*****************************************************************************/
```

```
int faktorisiere_N(N,teiler1, teiler2)
     LONGTYPE  N, teiler1, teiler2;

{
  FILE          *fp;
  LONGTYPE      D, HILFSVAR, kN;
  register int k, p, i, rcode, testwert, zaehl_anzahl=0;
  int           f1, f2, f3, st, art, groesse_FB, M, small, part, pp, anzahl;
  int           zeit[2],stellenzahl, gesamt, vergleich_cluster = 0;
  int           *SQRTkN = NULL, *FB = NULL;
  char          dstart[50], dende[50], zeile[100], command[200], fakt[40];
  char          logfile[90], result_name[80], antwort[3], *gestartet, fertig = 0;
  double        T;

  kN = lalloc(stel(N) + 5);

/****************************************************************************/
/*   Bestimmung des Multiplier k                                            */
/*   Die Knuth-Schroeppelfunktion wird fuer BIS_PRIMZAHL viele Faktorbasis-   */
/*   elemente ausgewertet                                                   */
/****************************************************************************/

  if (( k = finde_k(N, BIS_PRIMZAHL)) < 0)
    fehler(k);

  printf("Multiplier k = %d\n", k);

  lsmul(N, k, kN);
  lstring(kN,zeile);
  stellenzahl = strlen(zeile);

/****************************************************************************/
/*   Speicherallokieren fuer Parameter der Polynomerzeugung D und eine       */
/*   Hilfsvariable                                                          */
/****************************************************************************/

  D         = lalloc(stel(kN)+1);
  HILFSVAR  = lalloc(stel(kN));

/****************************************************************************/
/*   Einlesen der Parameter ( Toleranzwert, Siebintervallaenge, Groesse der   */
/*   Faktorbasis ) fuer eine Zahl mit stellenanzahl Dezimalstellen          */
/****************************************************************************/

  if ((rcode = get_parameter(stellenzahl, &T, &M, &groesse_FB)) < 0)
    fehler(rcode);

  printf("Toleranz: %05.2lf\nFB: %d\nSiebgroesse : %d\n",T, groesse_FB,M);
  printf("Index der 1.Siebprimzahl:%d\n\n",kleinstart);
  fflush(stdout);

/****************************************************************************/
/*   Erstellen der Faktorbasis fuer kN                                      */
/****************************************************************************/

  if ((rcode = bestimme_FB(groesse_FB, kN, &FB,  k)) < 0)
    fehler(rcode);

/****************************************************************************/
/*   Verlasse die Prozedur falls ein Teiler beim Erstellen der Faktorbasis   */
/*   gefunden wurde.                                                        */
/****************************************************************************/

  if (rcode > 1)
    {
      slsto(rcode, teiler1);
```

```
      llsto(teiler1, teiler2);
      return(0);
    }

/*****************************************************************************/
/*   Berechnung des Testwertes mit dem jeder Siebeintrag nach dem Siebn      */
/*   verglichen wird.                                                        */
/*****************************************************************************/

  testwert = LOGMUL * (int)(0.5 * log2(ldouble(kN)) + log2((double)M) -
             T*log2((double)FB[FB[0]+1]));

/*****************************************************************************/
/*   Berechnung der Ressolwurzeln und Allokieren des benoetigten Speicherplatz*/
/*   Die Werte werden in einem Integer Feld SQRTkN abgespeichert und zum Ver- */
/*   schicken an die Clienten in die Datei RESSOLW abgespeichert.            */
/*****************************************************************************/

  if (!(SQRTkN = (int *)calloc (groesse_FB + 3 , sizeof(int))))
    fehler(-1110);

  for (i = 2; i <= groesse_FB + 1; i++)
    {
      p = FB[i];
      if ((SQRTkN[i] = m_ressol(lsmod(kN, p), p)) < 0)
        SQRTkN[i] += p;
    }
  if (!(fp=fopen("RESSOLW","w")))
    fehler(-2323);
  for (i=2; i <= groesse_FB + 2 ;i++)
    fprintf(fp,"%d\n",SQRTkN[i]);
  fclose(fp);

/*****************************************************************************/
/*   Test ob das Programm aufgesetzt wurde ( dann existiert eine Datei LAST_D */
/*   mit dem letzten verschickten Wert fuer D). Sonst wird der Wert fuer     */
/*   STARTD ( = ((Sqrt(kN/2)/M) berechnet. Ausserdem wird sichergestellt, dass*/
/*   STARTD kongruent 3 modulo 4 ist.                                        */
/*****************************************************************************/

  if (fp=fopen("LAST_D","r"))
    {
      flread2(fp,D);
      fclose(fp);
      printf("Program wurde wiederaufgesetzt. Last_D = ");
      lwrite(D);
      fflush(stdout);
    }
  else
    {
      lsdiv(kN, 2, HILFSVAR, NULL);
      longsqrt(HILFSVAR, HILFSVAR);
      lsdiv(HILFSVAR, M, HILFSVAR, NULL);
      longsqrt(HILFSVAR, D);
      if (!lscomp(D, 0))
        slsto(3, D);
      D[1] = D[1] | 3;
      lsadd(D,-4,D);
    }

/*****************************************************************************/
/*  Bereitstellen der Dateien zur Speicherung der Ergebnisse                 */
/*  fuer full relations               FAKTORENMATRIX                         */
/*  fuer partial relations            PARTION                                */
/*  fuer partial-partial relations    PART.LARGE                             */
/*****************************************************************************/
```

```
  if (!(fp = fopen("FAKTORENMATRIX","a")))
    return(-2106);
  fclose(fp);
  if (!(fp = fopen("PARTION","a")))
    return(-2106);
  fclose(fp);
  if (!(fp = fopen("PART.LARGE","a")))
    return(-2106);
  fclose(fp);

/*****************************************************************************/
/*  Starten von LIPS und den verfuegbaren Clients                            */
/*  rmachine_number wird vom Lipssytem gesetzt und enthaelt die Anzahl der   */
/*                  Clients in der Client_Conf                               */
/*****************************************************************************/

  printf("Initlips : %d\n",initlips("CLCONF", NULL, NULL));
  if (rmachine_number < 1)
    {
      printf("Es stehen Remote-Rechner zur Verfuegung !!!\n");
      exit(1);
    }
  printf("Es wurden %d Rechen gestartet.\n",rmachine_number);fflush(stdout);

/*****************************************************************************/
/* Fuer jeden Client wird ein Eintrag im Feld gestartet angelegt.            */
/* Konnte der Rechner i gestartet werden so ist gestartet[i] = 0             */
/* sonst ist der Wert -1                                                     */
/*****************************************************************************/

  if (!(gestartet = calloc (rmachine_number + 3 , sizeof(char))))
    fehler(-2107);

/*****************************************************************************/
/* Starten der einzelnen Rechner und fuehren des Eintrags in gestartet       */
/* Die Struktur machine[i] wird vom Lips-system zur Verfuegung gestellt.     */
/* Sie enthaelt als Komponente den Name des Rechners (name) und eine Cluster- */
/* nummer (cluster), die die Zugehoerigkeit zu einem Filesystems angeben     */
/*****************************************************************************/

  for (i=0; i<rmachine_number;i++)
    {
      sprintf(logfile,"LOGFILE.%s",machine[i].name);
      if ((gestartet[i] = rstart("PMPQS-worker",machine[i].name,logfile)) != 0)
        l_error("Starten des Rechners : ");
      printf("rstart von %s : %d\n",machine[i].name,gestartet[i]);
      fflush(stdout);
    }

/*****************************************************************************/
/* An alle gestarteten Rechner wird die Faktorbasis, die Ressolwurzeln, die   */
/* Konfigurationsdatei und eine Fehlertabelle verschickt. Vorher wird         */
/* getestet, ob die Dateien bereits zu einem Rechner im gleichen Cluster      */
/* geschickt wurden.                                                          */
/*****************************************************************************/

  vergleich_cluster = 0;

  for (i=0; i<rmachine_number ;i++)
    {
      if (gestartet[i] == 0)
        {
          if (vergleich_cluster != machine[i].cluster)
            {
              if ((f1 = rcp(machine[i].name,"FAKTORBASIS",0)) != 0)
```

```
              l_error(" Fehler bei Faktorbasis verschicken : ");
            if ((f2 = rcp(machine[i].name,"RESSOLW",0)) != 0)
              l_error(" Fehler bei Ressolwurzel verschicken : ");
            if ((f3 = rcp(machine[i].name,"FEHLER_TAB",0)) != 0)
              l_error(" Fehler bei Fehlertabelle verschicken : ");
            if (!(f1|| f2 ||f3 ))
              {
                vergleich_cluster = machine[i].cluster;
                printf("DATEIEN an MASCHINE %s verschickt\n",machine[i].name);
              }
          }

/****************************************************************************/
/* Zum Verschicken von D-Intervallen wird D in einen String dstart umgewandelt*/
/* die Intervalllaenge (10000) aufaddiert und dann in den String dende umge-  */
/* wandelt                                                                    */
/****************************************************************************/

          lstring(D,dstart);
          lsadd(D,10000,D);
          lstring(D,dende);
          lsadd(D,4,D);

/****************************************************************************/
/* Dem Rechner i wird eine Botschaft mit den benoetigten Parametern geschickt */
/* Diese dient als Startbotschaft fuer die Clients.                           */
/****************************************************************************/


          if (out(machine[i].name,"START","iiiiisss",M,testwert,stel(kN),SIEBINT
                  ERN,groesse_FB,dstart,dende,machine[i].name))
            l_error("Fehler bei OUT von START : ");
          printf("D's an MASCHINE %s verschickt ( %s - %s)\n",machine[i].name,
                  dstart, dende);
          fflush(stdout);
        }
    }
  fclose(fp);


/****************************************************************************/
/* Hier beginnt die Schleife, in der der Master mit den Clients kommuniziert  */
/* Diese wird durchlaufen, bis genuegend Relationen gefunden wurden, um das   */
/* lineare Gleichungssytem zu loesen.                                         */
/****************************************************************************/

  while (1)
    {

/****************************************************************************/
/*  Aus der Ergebnisdatei wird eingelesen wieviele Relationen bisher gefunden */
/*  wurden. Sind genuegend Relationen gefunden worden, dann wird das Netz     */
/*  abgebaut, und das Programm zum Erzeugen der Eingabedatei fuer das Gauss-  */
/*  -Programm wird aufgerufen.                                                */
/****************************************************************************/

      if (fp=fopen("/home/denny/erg","r"))
        {
          while (fgets(zeile,100,fp));
          fclose(fp);
          sscanf(zeile,"%d %*d %*d %*d %d",&small , &part);
          if (small + part > groesse_FB)
            {
              kill_all_clients();
              exitlips();
              auswahl();
```

```
            zykelfind( FB, kN, k);
            gausvor();
            exit();
          }
        else
          {
            printf("Es wurden %d Relationen gefunden\n",small+part);
            fflush(stdout);
          }
   }
/****************************************************************************/
/* Es wird auf eine Botschaft eines Clients gewartet. Die Variable art unter- */
/* scheidet zwischen einer Rueckmeldung mit Bitte um eine neues Intervall (=1)*/
/* und der Bestaetigung, dass die Ergebnisdateien uebertragen wurden (= 2)    */
/****************************************************************************/

   if (in("MSG","ISIIIIII",&art,result_name,&small,&part,&anzahl,&pp,&zeit[0]
       ,&zeit[1]))
     l_error("Fehler bei IN  : ");
   switch(art)
     {
       case 1:

/****************************************************************************/
/* Meldung 1: Ein neues Startintervall wird verschickt und die Ergebisse      */
/*            werden protokolliert.                                           */
/****************************************************************************/

        lstring(D,dstart);
        lsadd(D,10000,D);
        lstring(D,dende);
        lsadd(D,4,D);
        fp=fopen("LAST_D","w") ;
        flwrite2(fp,D);
        fclose(fp);
        system("date");
        printf("%s  %d RELATIONS  %d PARTIONS ",result_name, small, part);
        printf("%d DOUBLE  %d D zeit: %d  %d zaehl %d\n",pp, anzahl,zeit[0],
                zeit[1],zaehl_anzahl);
        fflush(stdout);
        zaehl_anzahl += anzahl;
        if (out(result_name,"O.K.","sss",result_name,dstart,dende))
          l_error("Fehler bei OUT von O.K  : ");
        printf("D's an MASCHINE %s verschickt ( %s - %s)\n",result_name,
                dstart, dende);fflush(stdout);
        break;
       case 2:

/****************************************************************************/
/* Meldung 2: Die vom Client gesendeten Ergebisdateien werden uebernommen     */
/****************************************************************************/

        printf("Uebernehme Ergebnisse von %s\n",result_name);
        fflush(stdout);

        sprintf(fakt,"%s.%s","FAKTORENMATRIX",result_name);
        sprintf(command,"cat %s >> %s",fakt,"FAKTORENMATRIX");
        system(command);
        unlink(fakt);

        sprintf(fakt,"%s.%s","PARTION",result_name);
        sprintf(command,"cat %s >> %s ",fakt,"PARTION");
        system(command);
        unlink(fakt);

        sprintf(fakt,"%s.%s","PART.LARGE",result_name);
```

```
        sprintf(command,"cat %s >> %s ",fakt,"PART.LARGE");
        system(command);
        unlink(fakt);

/*****************************************************************************/
/* Nachdem 30000 Polynome abgearbeitet wurden, werden die neuen Ergebnis -    */
/* dateien mit den bereits gesammelten verknuepft und die Anzahl der          */
/* gefundenen Relationen ermittelt und die Ergebnisdatei geplegt.             */
/* Damit waehrend dieser Zeit der Master nicht blockiert wird, ruft das       */
/* Programm eine Shell und startet das Zusammenschreiben dort.                */
/*****************************************************************************/

        if (zaehl_anzahl >30000)
          {
            zaehl_anzahl = 0;
            system("mv FAKTORENMATRIX DATEN/. ; mv PART.LARGE DATEN/. ; mv
                    PARTION DATEN/. ; cd DATEN ; zusam >>/home/denny/erg &");
          }
      }
  }
  free(kN);
  free(D);
  free(HILFSVAR);
  return(rcode);
}
```

best_FB.c

```
#include "hmpqs.h"

/*****************************************************************************/
/* DOC                 BESTIMME_FB()                                         */
/*                                                                           */
/*      Die Funktion erstellt die Faktorbasis. Es wird zuerst ueberprueft, ob */
/*      zu der zu faktorisierenden Zahl eine FB in ausreichender Groesse     */
/*      vorhanden ist. Ist keine FB als Datei vorhanden, so wird sie neu     */
/*      erstellt. Es werden alle Primzahlen in die FB aufgenommen, fuer die  */
/*      das Legendre-Symbol (N|p) = 1 ist (bis genuegend Elemente in der     */
/*      Faktorbasis sind) Ist eine FB fuer N vorhanden so wird sie bei       */
/*      richtiger Groesse uebernommen, sonst bei zu grosser Groesse          */
/*      verkleinert und bei zu kleiner Groesse erweitert.                    */
/*                                                                           */
/*      Eingabe:  int       groesse    = Groesse der FB                      */
/*                int       k          = Multiplier                          */
/*                LONGTYPE  kN         = zu fakt. Zahl                       */
/*                                                                           */
/*                                                                           */
/*      Ausgabe:  int **FB        = Faktorbasis                              */
/*                                                                           */
/*      Return :  1              Faktorbasis korrekt erstellt                */
/*                Fehlernummer   (immer < 0)                                 */
/*                Teiler von N   sonst                                       */
/*                                                                           */
/*     zuletzt geaendert am 09.04.92                                         */
/*****************************************************************************/
int bestimme_FB(groesse, kN, FB, k)
     LONGTYPE       kN;
     register int   groesse, **FB, k;
{
  FILE            *fp;
  int             old_size;
  register int    p, num = 0, teiler = 1, i;
  LONGTYPE        old_N;
```

```
  if (*FB)
    free(*FB);
  if (!(*FB = (int *)calloc (groesse + 3 , sizeof(int))))
    return(-3);

/***************************************************************************/
/* Existiert bereits eine Datei Faktorbasis ?                              */
/* Falls ja lese Zahl ein, fuer die die Faktorbasis erstellt wurde         */
/***************************************************************************/

  if (fp = fopen("FAKTORBASIS", "r"))
    {
      fscanf(fp, "%d", &old_size);
      old_N = lalloc(DEFAULT_LENGTH);
      flread2(fp, old_N);
      if (!llcomp(kN, old_N))
        {
          printf("Benutze alte Faktorbasis \n");
          fflush(stdout);

/***************************************************************************/
/* Einlesen der Faktorbasis                                                */
/***************************************************************************/

          *FB++ = groesse;
          while (groesse-- >= 0)
            fscanf(fp, "%d", FB++);
          *FB = 0;
          fclose(fp);

          if (old_size > groesse)
            {
              printf("Kuerze Faktorbasis\n");
              fflush(stdout);
              if (!(fp = fopen("FAKTORBASIS", "w")))
                return(-2201);
              abspeichern(*FB, kN, groesse, fp);
             }
          if (old_size < groesse)
            {
              printf("Erweitere Faktorbasis\n");
              fflush(stdout);
              teiler = erweitern(*FB, kN, old_size, groesse, k);
              if (teiler < 0)
                {
                  free(old_N);
                  return(teiler);
                }
              if (!(fp = fopen("FAKTORBASIS", "w")))
                return(-2201);
              abspeichern(*FB, kN, groesse, fp);
            }
        }
      else
        {
          fclose(fp);
          teiler = erweitern(*FB, kN, 0, groesse, k);
          if (!(fp = fopen("FAKTORBASIS", "w")))
            return(-2201);
          abspeichern(*FB, kN, groesse, fp);
        }
    }
  else
    {
      teiler = erweitern(*FB, kN, 0, groesse, k);
```

```
      if (teiler < 0)
        {
          free(old_N);
          return(teiler);
        }
      if (!(fp = fopen("FAKTORBASIS", "w")))
        return(-2201);
      abspeichern(*FB, kN, groesse, fp);
    }
  fclose(fp);
  free(old_N);
  return(teiler);
}

/****************************************************************************/
/* FUNKTION: erweitern(FB, kN, groesse, fp)                                 */
/*                                                                          */
/* Die Funktion erweitert eine bestehende FB oder erstellt eine neu neue FB */
/* (=erweitern fuer alte groesse 0)                                         */
/*                                                                          */
/*      Eingabe:  int  old_size  = Groesse der alten FB                     */
/*                int  groesse   = Groesse der neuen FB                     */
/*                LONGTYPE  kN   = zu fakt. Zahl N                          */
/*                                                                          */
/*      Ausgabe:  int  *FB       = erstellte FB                             */
/****************************************************************************/
int erweitern(FB, kN, old_size, groesse, k)
     int        *FB, old_size, groesse, k;
     LONGTYPE   kN;
{
  register int     maxprime = (old_size == 0 ? 0 : FB[old_size + 1]);
  register int     teiler = 1;
  FILE             *fp = fopen("/usr/local/dat/PRIMES", "r");
  int p = 0;

  if (!fp)
    return(-2104);
  FB[0] = groesse;
  FB[1] = -1;
  FB = &FB[old_size + 2];

  while (p != maxprime)
    fscanf(fp, "%d", &p);

  while (old_size < groesse)
    {
      if (fscanf(fp, "%d", &p) == EOF)
        return(-321);

      switch (legendre(kN,p))
        {
          case 1:
                 *FB++ = p;
                 old_size++;
                 break;
          case 0:
                 if (k%p == 0)
                   {
                     *FB++ = p;
                     old_size++;
                   }
                 else
                   teiler = p;
                 break;
         }
    }
```

```
  *FB = 0;
  return(teiler);
}

/*****************************************************************************/
/* FUNKTION: erweitern(FB, kN, groesse, fp)                                  */
/*                                                                           */
/*      Die Funktion speichert eine erstellt oder erweiterte Faktorbasis in   */
/*      eine Datei ab. In der ersten Zeile steht die Groesse der FB (ohne -1).*/
/*      In der zweiten Zeile steht die zu fakt. Zahl N. In der dritten Zeile  */
/*      steht die -1 fuers Vorzeichen. Ab der 4.ten Zeile folgen die eigent-  */
/*      lichen FB-Elemente. In der letzten Zeile steht eine 0.                */
/*                                                                           */
/*      Eingabe:  int       *FB       = Faktorbasis                          */
/*                LONGTYPE  *kN       = zu fakt. Zahl N                      */
/*                int       groesse   = Groesse der FB                       */
/*                FILE      *fp       = Pointer auf Dateianfang              */
/*****************************************************************************/
abspeichern(FB, kN, groesse, fp)
     int      *FB, groesse;
     LONGTYPE kN;
{
  fprintf(fp, "%d\n", groesse);
  flwrite2(fp, kN);
  while (*++FB)
    fprintf(fp, "%d\n", *FB);
  fprintf(fp, "0\n");
}
```

auswahl.c

```
#include"hmpqs.h"
#define S 3000000

int      *ZDAT;
char     *ANZAHL;
/*****************************************************************************/
/* DOC                          auswahl()                                    */
/*                                                                           */
/*      Die Funktion sucht diejenigen partial and partial-partial Relationen  */
/*      die large primes enthalten , die insgesamt nur einmal auftreten       */
/*      Dazu werden zuerst alle Relationen eingelesen und fuer jede large     */
/*      Prime die Anzahl ihres Auftretens bestimmt.                           */
/*      Dann folgt der 1. Eliminationsschritt. Man sucht die Relationen,      */
/*      die nur large primes haben, die mehr als einmal auftauchen.           */
/*      Danach wird ein 2.Eliminationsschritt durchgefuehrt, da sich die      */
/*      Anzahl des Auftretens von large primes durch weglassen nutzloser      */
/*      Relationen veraendert  verringert.                                    */
/*      Return :  %                                                           */
/*                                                                           */
/*      zuletzt geaendert am 20.03.93                                         */
/*****************************************************************************/

auswahl()
{
  FILE          *fpin, *fp ;
  int           p[2],j[2];
  char          zeile[2000];
  register int  i, ii=0;
```

```
/****************************************************************************/
/* Allokieren der Hashtafel und Vorbesetzen der Eintraege                  */
/****************************************************************************/

 if (!(ZDAT = (int *)malloc (S * sizeof(int))))
    return(-12);
  if (!(ANZAHL = calloc (S , sizeof(char))))
    return(-13);

  memset(ZDAT, -1 ,S * sizeof(int));

/****************************************************************************/
/*  Einlesen der Partial-Relationen + zaehlen der Vorkommen von Primzahlen    */
/****************************************************************************/

  sprintf(zeile,"DATEN/PART.%d",ii++);

  while (fpin = fopen(zeile, "r"))
    {
      while(fgets(zeile,2000,fpin))
        {
          sscanf(zeile,"%d ",&p[1]);
          zaehle(p[1]);
        }
      fclose(fpin);
      sprintf(zeile,"DATEN/PART.%d",ii++);
    }

/****************************************************************************/
/*  Einlesen der PPartial-Relationen + zaehlen der Vorkommen von Primzahlen   */
/****************************************************************************/

  ii = 0;
  sprintf(zeile,"DATEN/DOUBLE.%d",ii++);
  while (fpin = fopen(zeile, "r"))
    {
      while(fgets(zeile,2000,fpin))
        {
          sscanf(zeile,"%d %d",&p[0],&p[1]);
          zaehle(p[0]);
          zaehle(p[1]);
        }
      fclose(fpin);
      sprintf(zeile,"DATEN/DOUBLE.%d",ii++);
    }
  fclose(fpin);

/****************************************************************************/
/*  Einlesen der Partial-Relationen, falls eine sicher nicht in einem Zykel   */
/*  vorkommt wird die partial-Relation nicht in ZYKEL.FIND abgespeichert.     */
/****************************************************************************/

  fp = fopen ("ZYKEL.FIND","w");
  ii = 0;
  sprintf(zeile,"DATEN/PART.%d",ii++);

  while (fpin = fopen(zeile, "r"))
    {
      while(fgets(zeile,2000,fpin))
        {
```

```
          sscanf(zeile,"%d ",&p[1]);
          j[1] = p[1] % S;

          while ((ZDAT[j[1]] != -1) && (ZDAT[j[1]] !=p[1]))
            j[1] = (j[1] < S-1)?j[1]+1:0;

          if (ANZAHL[j[1]] > 1)
            fprintf(fp,"1 %s",zeile);
          else
            ANZAHL[j[1]]= 0;
         }
      fclose(fpin);
      sprintf(zeile,"DATEN/PART.%d",ii++);
    }
  ii = 0;

/****************************************************************************/
/*  Einlesen der PPartial-Relationen, falls eine pp-Relation sicher nicht in  */
/*  einem Zykel vor kommt (eine Primzahl insgesamt nur 1 mal) wird sie nicht  */
/*  abgespeichert in ZYKEL.FIND . Um moeglichst viele Relationen zu aus-      */
/*  schliessen zu koennen sollte man mit den pp-Relation anfangen. Dies       */
/*  wuerde jedoch zu viel zu langen Zykeln beim Zykelfinden fuehren, da dann  */
/*  dort zuerst die pp-Relationen eingelsen wuerden.                          */
/****************************************************************************/

  sprintf(zeile,"DATEN/DOUBLE.%d",ii++);
  while (fpin = fopen(zeile, "r"))
    {
      while(fgets(zeile,2000,fpin))
        {
          sscanf(zeile,"%d %d",&p[0],&p[1]);
          for (i=0;i<2;i++)
            {
              j[i]= p[i] % S;
              while ((ZDAT[j[i]] != -1) && (ZDAT[j[i]] != p[i]))
               j[i] = (j[i] < S-1)?j[i]+1:0;
            }
          if ((ANZAHL[j[0]] > 1) && (ANZAHL[j[1]] > 1))
            fprintf(fp,"%s",zeile);
          else
           {
             ANZAHL[j[0]]--;
             ANZAHL[j[1]]--;
           }
        }
      fclose(fpin);
      sprintf(zeile,"DATEN/DOUBLE.%d",ii++);
    }
  fclose(fp);
  fclose(fpin);

/****************************************************************************/
/* Hier beginnt der 2. Lauf, der das Ergebnis des 1.Laufs erneut ueberprueft  */
/* ob nicht benoetigte Relationen in der Datei ZYKEL.FIND sind                */
/* Ergebnis wird in das File LAUF.2  geschrieben                              */
/****************************************************************************/

  fpin = fopen ("ZYKEL.FIND","r");
  fp = fopen ("LAUF.2","w");
  while(fgets(zeile,2000,fpin))
```

```
    {
      sscanf(zeile,"%d %d",&p[0],&p[1]);
      if (p[0] == 1)
        {
         j[1] = p[1] % S;
          while ((ZDAT[j[1]] != -1) && (ZDAT[j[1]] !=p[1]))
            j[1] = (j[1] < S-1)?j[1]+1:0;
          if (ANZAHL[j[1]] > 1)
            fprintf(fp,"%s",zeile);
          else
            ANZAHL[j[1]]= 0;
        }
      else
        {
          for (i=0;i<2;i++)
            {
              j[i]= p[i] % S;
              while ((ZDAT[j[i]] != -1) && (ZDAT[j[i]] != p[i]))
                j[i] = (j[i] < S-1)?j[i]+1:0;
            }
          if ((ANZAHL[j[0]] > 1) && (ANZAHL[j[1]] > 1))
            fprintf(fp,"%s",zeile);
          else
            {
              ANZAHL[j[0]]--;
              ANZAHL[j[1]]--;
            }
        }

     }
  fclose(fp);
  fclose(fpin);
  unlink("DATEN/REF* DATEN/PART.* DATEN/DOUBLE.* DATEN/SMALL ZYKEL.FIND");
  free(ZDAT);
  free(ANZAHL);

}

/*****************************************************************************/
/* FUNKTION : zaehle                                                         */
/* Sucht den Hasheintrag einer large prime p und inkrementiert deren Zaehler  */
/*****************************************************************************/

zaehle ( p )
     int p;
{
  register int j;

  j = p % S;

  while ((ZDAT[j] != -1) && (ZDAT[j] !=p))
    j = (j < S-1)?j+1:0;

  ANZAHL[j]++;

  if (ZDAT[j] == -1)
    ZDAT[j] = p;
}
```

zykelfind.c

```
#include"hmpqs.h"
#define S 1200000  /* HASH-TAFEL Groesse */

int         *DAT,*VERWEIS,*REL;
static short *exponenten = NULL;

/****************************************************************************/
/* DOC                          ZYKELFIND                                   */
/*                                                                          */
/*      Die Funktion ermittelt, aus einer Menge von partial und partial-    */
/*      partial Relationen, elementare Zykel die zu full relations fuehren. */
/*      Dabei wird eine Art Union-Find Algorithmus verwendet. Der Algorithmus */
/*      findet dabei nicht die kuerzesten Zykel. Dies stellt jedoch keinen  */
/*      grosen Nachteil fuer das Loesen des LGS da, und Algorithmen die     */
/*      kuerzeste Zykel finden haben einen sehr hohen Platz- und Zeitbedarf. */
/*      Die Reltionen werden aus der Datei LAUF.2, die in der Funktion      */
/*      auswahl erzeugt wird, gelesen und die kombinierten Relationen werden */
/*      an die Dateo SMALL angehaengt.                                      */
/*                                                                          */
/*      Eingabe:  LONGTYPE kN         = zu fakt. Zahl mit Multipl.          */
/*                int *FB             = Faktorbasis                         */
/*                int k               = Multiplier                          */
/*                                                                          */
/*      Return :  %                                                         */
/*                                                                          */
/*      zuletzt geaendert am 20.03.93                                       */
/****************************************************************************/
zykelfind (FB, kN, k)
     int *FB, k;
     LONGTYPE kN;

{
  int            stell,start,p[2],maximum,r[2],j[2],zaehl[40], p1,p2;
  FILE           *fakneu = fopen("SMALL","a"), *fpin = fopen("LAUF.2","r");
  char           *faktor, zeile[2000], h1_string[150],h2_string[150];
  register int   i,flag, oft;
  long           sicher, wo;
  LONGTYPE       Fbh, H1, H2, TEMP;

  lsdiv(kN,k,kN,0);

/****************************************************************************/
/* Allokieren von Speicher fuer lange Zahlen und char-Vektoren              */
/****************************************************************************/

  Fbh   = lalloc(4);
  H1    = lalloc(30);
  H2    = lalloc(30);
  TEMP  = lalloc(30);

  if (!(faktor = (char *)malloc(300 * sizeof(char))))
    fehler(-2214);
  if ((exponenten = (short*) malloc((10 + FB[0]) * sizeof(short))) == NULL)
    fehler(-2215);

/****************************************************************************/
/* Allokieren der Hashtafel und Vorbesetzen der Eintraege                   */
/****************************************************************************/

  if (!(REL = (int *)malloc (S * sizeof(int))))
    return(-11);
  if (!(DAT = (int *)malloc (S * sizeof(int))))
    return(-12);
```

```
  if (!(VERWEIS = (int *)malloc (S * sizeof(int))))
    return(-13);
  memset(DAT, -1 ,S * sizeof(int));
  memset(REL, 0 ,S * sizeof(int));
  wo = 0;
  while(fgets(zeile,2000,fpin))
    {
      sscanf(zeile,"%d %d",&p[0],&p[1]);

/****************************************************************************/
/* Bestimmung der Komponenten in denen beide large primes liegen.          */
/****************************************************************************/

      for (i=0;i<2;i++)
        {
          j[i] = p[i] % S;
          while ((DAT[j[i]] != -1) && (DAT[j[i]] !=p[i]))
            j[i] = (j[i] < S-1)?j[i]+1:0;
          if (DAT[j[i]] == -1)
            {
              DAT[j[i]] = p[i];
              VERWEIS[j[i]] = j[i];
            }
          r[i] = VERWEIS[j[i]];
          while ( r[i] != VERWEIS[r[i]])
             r[i] = VERWEIS[r[i]];
        }

/****************************************************************************/
/* Liegen beide Zahlen in der selben Komponente ?                          */
/* Falls ja wird ein Zykel geschlossen und die Relation aufgebaut und getestet*/
/* sonst wird eine Kante fuer die Relation aufgenommen.                     */
/****************************************************************************/

      if (r[0] != r[1])
        {
          maximum = (DAT[r[0]] < DAT[r[1]]) ? 1 : 0 ;
          if ( r[maximum] != j[maximum])
            reverse( j[maximum],r[maximum]);

          VERWEIS[j[maximum]] = j[1-maximum];
          REL[j[maximum]] = wo;
        }
      else
        {
           memset(exponenten, 0, sizeof(short) * (FB[0]+10));
           sscanf(zeile, "%d %d @ %s",&p1,&p2, h1_string);
           string_to_l(h1_string,H1);
           set1(H2);
           addexp(zeile);
           oft = 1;
           flag= 0;
           sicher = ftell(fpin);
           reverse(j[0], r[1]);
           start = j[1];
           while (start != j[0])
             {
               oft++;
               fseek(fpin,REL[start],0);
               fgets(zeile,2000,fpin);
               sscanf(zeile, "%d %d @ %s", &p1,&p2, h1_string);
               string_to_l(h1_string,TEMP);
               if (flag)
                 {
                   addexp(zeile);
                   llmul(H1,TEMP,H1);
```

```
                lldiv(H1,kN,0,H1);
              }
            else
              {
                subexp(zeile);
                llmul(H2,TEMP,H2);
                lldiv(H2,kN,0,H2);
              }
            flag = !flag;
            start = VERWEIS[start];
          }
        fseek(fpin,sicher,0);
        zaehl[oft]++;
        if (!flag)
          {
            lsmul(H2,DAT[j[0]],H2);
            lldiv(H2,kN,0,H2);
          }
        lstring(H1,h1_string);
        lstring(H2,h2_string);
        reverse(r[1],j[0]);
        VERWEIS[r[1]] = r[1];

        lsquare(H1,H1);
        lldiv(H1, kN, 0, H1);
        lsquare(H2,H2);
        lldiv(H2, kN, 0, H2);
        memset(faktor, 0, 300);

/****************************************************************************/
/* Lese aus den Vektor exponenten die Exponenten der gefunden Relation      */
/* Baut in dem String faktor eine Darstellung der Zerlegung auf.            */
/* Ausserdem wird die rechte und linke Seite der Relation berechnet.        */
/****************************************************************************/

        for (i = 1; i <= FB[0] + 1 ; i++)
          {
            if (exponenten[i])
              {
                sprintf(faktor,"%s %d %d",faktor,exponenten[i],i);
                slsto(FB[i],Fbh);
                set1(TEMP);
                stell = abs(exponenten[i]);
                while (stell-- != 0)
                  llmul(TEMP, Fbh, TEMP);

                if (exponenten[i] < 0)
                  {
                    llmul(H1, TEMP, H1);
                    lldiv(H1, kN, 0, H1);
                  }
                else
                  {
                    llmul(H2, TEMP, H2);
                    lldiv(H2, kN, 0, H2);
                  }
          }
        }
      if (lsig(H2) == -1)
        lladd(kN,H2,H2);
      if (lsig(H1) == -1)
        lladd(kN,H1,H1);

/****************************************************************************/
/* Wurde eine Kongruenz gefunden ? Falls ja speichere sie in SMALL          */
/****************************************************************************/
```

```
         if ((llcomp(H2,H1) == 0) )
           fprintf(fakneu, "%s %s @:%s 0\n",h1_string, h2_string,faktor);
        }
        wo = ftell(fpin);
    }
  fclose(fpin);
  fclose(fakneu);

/****************************************************************************/
/* Schreibe Statistik ueber Laenge der Zykel in Datei ZYKEL                 */
/****************************************************************************/

  fpin = fopen("ZYKEL","w");
  for(i=2;i<40;i++)
    if (zaehl[i])
      fprintf(fpin,"Zykel der Laenge %d : %d \n",i,zaehl[i]);
  fclose(fpin);
}
/****************************************************************************/
/* FUNKTION : reverse (i,j)                                                 */
/*                                                                          */
/* Die Funktion laeuft im Graph von Knoten i nach j und aendert die Orientier-*/
/* ung der Kanten.                                                          */
/****************************************************************************/

reverse ( i , j)
      register int i,j;
{
  register int hilfverweis, weiter, next, relation1, relation;

  hilfverweis   = i;
  weiter         = VERWEIS[i];
  next           = VERWEIS[weiter];
  relation       = REL[weiter];
  VERWEIS[weiter] = hilfverweis;
  REL[weiter]     = REL[hilfverweis];

  while (weiter != j)
    {
      hilfverweis     = weiter;
      weiter          = next;
      next            = VERWEIS[weiter];
      VERWEIS[weiter] = hilfverweis;
      relation1       = REL[weiter];
      REL[weiter]     = relation;
      relation        = relation1;
    }
}

/****************************************************************************/
/* FUNKTION : addexp (s)                                                    */
/*                                                                          */
/* Die Funktion liest aus den Sring s eine Zerlegung und addiert die        */
/* Exponenten der Zerlegung zu den bereits in exponenten gespeicherten dazu. */
/* Entspricht der Multiplikation von rechten Seiten von Relationen.         */
/****************************************************************************/

int addexp(s)
     char    *s;
{
  char   *pp, *p, hilf[2000];
  int    e;

  strcpy(hilf,s);
```

```
  if (pp = strchr(hilf, ':'))
    {
      pp += 2;

      p = strtok(pp, " \n");
      while (p != NULL)
        {
          e = atoi(p);
          if (!e)
            break;
          p = strtok(NULL, " \n");
          exponenten[atoi(p)] += e;
          p = strtok(NULL, " \n");
        }
    }
}

/****************************************************************************/
/* FUNKTION : subexp (s)                                                    */
/*                                                                          */
/* Die Funktion liest aus den Sring s eine Zerlegung und subtrahiert die    */
/* Exponenten der Zerlegung zu den bereits in exponenten gespeicherten dazu. */
/* Entspricht der Division von rechten Seiten von Relationen.               */
/****************************************************************************/

int  subexp(s)
     char   *s;
{
  char   *pp, *p, hilf[2000];
  int    e;

  strcpy(hilf,s);

  if (pp = strchr(hilf, ':'))
    {
      pp += 2;

      p = strtok(pp, " \n");
      while (p != NULL)
        {
          e = atoi(p);
          if (!e)
            break;
          p = strtok(NULL, " \n");
          exponenten[atoi(p)] -= e;
          p = strtok(NULL, " \n");
        }
    }
}
```

gausvor.c

```
#include "hmpqs.h"

/****************************************************************************/
/* DOC           GAUSVOR                                                    */
/*                                                                          */
/* Die Funktion speichert die Datei SMALL in die fuer den Lenstra-Gauss-    */
/* Algorithmus notwendige Darstellung um.                                   */
/*                                                                          */
/*    zuletzt geaendert am 03.02.93                                         */
/****************************************************************************/
```

```
gausvor()
{
  FILE *fpin  = fopen("SMALL", "r");
  FILE *fpout = fopen("RELATIONS", "w");
  char *p, *pp, zeile[3000];
  register int i = 0, hil, size;

  while (fgets(zeile, 3000, fpin))
    {
      if (pp = strchr(zeile, ':'))
        {
          pp += 2;
          hil = laenge(pp, ' ');
          if ((hil & 1) == 1)
             hil--;

/*****************************************************************************/
/* schreibe den Zeilenanfang ( Nummer der Relation, Anzahl der Elemente)     */
/*****************************************************************************/

          fprintf(fpout, "%d %d 0", ++i, hil);
          p = strtok(pp, " \n");

/*****************************************************************************/
/* schreibe den Komponenten ( Exponent des Teilers , Index des Teilers)      */
/*****************************************************************************/

          while (p != NULL)
            {
              fprintf(fpout, " %d", abs(atoi(p)));
              p = strtok(NULL, " \n");
            }
          fprintf(fpout, "\n");
        }
    }
  fclose(fpin);
  fclose(fpout);
  return(0);
}

/*****************************************************************************/
/* FUNKTION : laenge (s,c)                                                   */
/*                                                                           */
/* Die Funktion liefert die Anzahl eines Zeichens c in einem String s        */
/*                                                                           */
/*****************************************************************************/

int laenge(s, c)
char *s, c;
{
    int l = 0;

    while (*s++)
        if (*s == c)
            l++;
    return(l);
}
```

A.1.2 Der Client

Das Makefile

```
################################################################################
#                                                                              #
#                            Makefile fuer WORKER                              #
#                                                                              #
################################################################################

VER=4

MACHINE=-sun$(VER)

INCLUDE=/home/denny/fopra/include
LIBDIR=/home/denny/lib
LIBDIR1=/home/berger/DIPLOM

LIBS=$(LIBDIR)/sieve.a $(LIBDIR1)/LARITH/larith.a
     /home/thsetz/lips_2.2F/lib/liblips.a

SLIPS=  -lm

OBJ=  worker.o ber_kor.o sieben.o pro_div.o gen_D.o

DEBUG=

OPT= -O4 -Bstatic
.KEEP_STATE:


PMPQS-worker: $(LIBS) $(OBJ)
cc $(OPT) $(MACHINE) -o PMPQS-worker $(OBJ) $(LIBS) $(SLIPS)

worker.o: worker.c
cc $(OPT) $(MACHINE) $(DEBUG) -I$(INCLUDE) -c worker.c

sieben.o: sieben.c
cc $(OPT) $(MACHINE) $(DEBUG) -I$(INCLUDE) -c sieben.c

gen_D.o: gen_D.c
cc $(OPT) $(MACHINE) $(DEBUG) -I$(INCLUDE) -c gen_D.c

pro_div.o: pro_div.c
cc $(OPT) $(MACHINE) $(DEBUG) -I$(INCLUDE) -c pro_div.c

ber_kor.o: ber_kor.c
cc $(OPT) $(MACHINE) $(DEBUG) -I$(INCLUDE) -c ber_kor.c
```

worker.c

```
#include "hmpqs.h"

short     small = 0 ,part = 0, pp = 0, anzahl=0;
int       timer[2], stell, SIEBSTART;
char      myname[64],smalldat[80], pdat[80], ppdat[80], dstart[70], dende[70];
LONGTYPE  MINLP,DOUBLE,LP,kN;
double    d_wurz;
```

```
/*****************************************************************************/
/* DOC                         WORKER                                        */
/*                                                                           */
/* Die Funktion versucht die Zahl N mit dem QS-Sieb-Algorithmus zu           */
/* faktorisieren. Sie dient als Aufruffunktion fuer die zur Faktorisierung   */
/* notwendigen Module.                                                       */
/*                                                                           */
/*  zuletzt geaendert am 03.02.92                                            */
/*****************************************************************************/

main()
{
  FILE          *fp;
  register int  p, i, rcode;
  int           testw, sieblae, groesse_FB; *SQRTkN = NULL, *FB = NULL;
  int           *START1 = NULL, *START2 = NULL, *CANDIDATE, send_ergeb();
  SIEBTYP       *sieb = NULL, *ende, *LOGP = NULL, testwert;
  LONGTYPE      D,DSTOP, A, B, D2inv;
  char          *faktor;

/*****************************************************************************/
/* Initialisieren der langen Arithmetik; Allokieren von Platz fuer lange     */
/* Zahlen                                                                    */
/*****************************************************************************/

  linit(DEFAULT_LENGTH);

  DOUBLE  = lalloc(6);
  MINLP   = lalloc(6);
  LP      = lalloc(6);

/*****************************************************************************/
/* Starten von Lips auf den CLient                                           */
/*****************************************************************************/

  initlips( NULL, NULL, NULL );

/*****************************************************************************/
/*  Warten auf Startmessage des Masters mit Parametern                       */
/*****************************************************************************/

  if ( in("START","IIIIISSS",&SIEBSTART, &testw, &stell, &sieblae, &groesse_FB,
          dstart,dende,myname) != 0)
    l_error("Fehler in StartIN :");

  printf("TESTWERT : %d \n#FB : %d\nstellen : %d \n",testw, groesse_FB, stell);
  printf("interne SIEBLAENGE : %d\nSIEBSTART : %d \n",sieblae, SIEBSTART);
  printf("Myname : %s\ndstart: %s\ndende: %s\n",myname, dstart, dende);
  fflush(stdout);

  timer[1] = timer[0] = 0;
  start_timing(timer);

  kN = lalloc(stell);
  testwert = (SIEBTYP) testw;

/*****************************************************************************/
/* Einlesen der Faktorbasis und der zu faktorisierenden Zahl                 */
/*****************************************************************************/

  if (!(FB = (int *)calloc (groesse_FB + 3 , sizeof(int))))
    fehler(-3456);
  if (!(fp  = fopen("FAKTORBASIS","r")))
    fehler(-999998);
  fscanf(fp, "%d", &groesse_FB);
  flread2(fp, kN);
```

```
  FB[0] = groesse_FB;

  for (i=1; i <= groesse_FB + 2 ;i++)
    fscanf(fp,"%d",&FB[i]);
  fclose(fp);

/****************************************************************************/
/* Speicherallokieren fuer lange Zahlen, Siebarray, Arrays und Vektoren     */
/****************************************************************************/

  D       = lalloc(2*stell);
  DSTOP   = lalloc(2*stell);
  A       = lalloc(2 * stell);
  B       = lalloc(2 * stell);
  D2inv   = lalloc(2 * stell);

  if (!(CANDIDATE = (int *)calloc(400,sizeof(int))))
    return(-1106);
  if (!(faktor = (char *)malloc(600 * sizeof(char))))
    return(-1107);
  if (!(sieb = (SIEBTYP *)malloc(sieblae * sizeof(SIEBTYP))))
    return(-1105);
  ende = sieb + sieblae - 1;


  LOGP    = (SIEBTYP *)calloc (groesse_FB + 3 , sizeof(SIEBTYP));
  SQRTkN  = (int *)calloc (groesse_FB + 3 , sizeof(int));
  START1  = (int *)calloc (groesse_FB + 3 , sizeof(int));
  START2  = (int *)calloc (groesse_FB + 3 , sizeof(int));

  if (!START1 || !START2 || !SQRTkN || !LOGP)
    fehler(-99999);

/****************************************************************************/
/* Einlesen der Ressolwurzeln aus der Datei RESSOLW                         */
/****************************************************************************/

  if (!(fp  = fopen("RESSOLW","r")))
    fehler(-999998);

  for (i=2; i <= groesse_FB + 2 ;i++)
    fscanf(fp,"%d",&SQRTkN[i]);
  fclose(fp);

/****************************************************************************/
/* Vorberechnung des "dualen" Logarithmus                                   */
/****************************************************************************/

 for (i=2; i <= groesse_FB + 1 ;i++)
   {
     p = FB[i];
     LOGP[i] = (SIEBTYP)(log2((double)p)*LOGMUL);
   }

  d_wurz = sqrt(ldouble(kN));

  slsto(LARGEP,LP);
  slsto(DOUBLEP,DOUBLE);
  lsquare(DOUBLE,DOUBLE);
  slsto(FB[groesse_FB+1],MINLP);
  lsquare(MINLP,MINLP);
  printf("LARGE : ");
  lwrite(LP);
  printf("DOUBLE : ");
  lwrite(DOUBLE);
  fflush(stdout);
```

```
  sprintf(smalldat,"%s.%s","FAKTORENMATRIX",myname);
  sprintf(pdat,"%s.%s","PARTION",myname);
  sprintf(ppdat,"%s.%s","PART.LARGE",myname);

/*****************************************************************************/
/* Ende der Vorberechnungen hier beginnt die Endlosschleife                  */
/*****************************************************************************/

  while (1)
    {
      small = part = pp = 0;
      string_to_l(dstart,D);
      string_to_l(dende,DSTOP);
      generiere_D(D);

/*****************************************************************************/
/* Ist das aktuelle Intervall abgearbeitet?                                  */
/*****************************************************************************/

      while (llcomp(DSTOP,D)!= -1)
        {
          if ((rcode = ber_koeffizienten(D, A, B, D2inv, FB, SQRTkN, START1,
                                  START2)) < 0)
            fehler(rcode);

          if ((rcode = sieben(FB, LOGP, START1, START2,testwert,sieb, CANDIDATE
                             ,sieblae, ende) < 0))
            fehler(rcode);

          if ((rcode = teste(A, B, D2inv, CANDIDATE, FB, START1, START2, faktor)
                            ) < 0)
            fehler(rcode);

          anzahl++;
          generiere_D(D);
        }
      send_ergeb();
    }
  free(D);
  free(A);
  free(B);
  free(D2inv);
}

/*****************************************************************************/
/* FUNKTION :  send_ergeb()                                                  */
/*                                                                           */
/* In der Funktion findet die Kommunikation mit dem Master statt. Zuerst     */
/* werden die statistischen Informationen verschickt (Message 1)             */
/* Nachdem der Master ein neues D-Intervall zurueckgeschickt hat beginnt     */
/* der Client mit dem Uebertragen der Ergebnisfiles. Ist diese beendet so    */
/* wird Message 2 verschickt und mit dem neuen Intervall weitergearbeitet    */
/*****************************************************************************/

int send_ergeb()
{
/*****************************************************************************/
/*  Einlesen der benoetigten Zeit                                            */
/*****************************************************************************/

  get_timing(timer);
  printf("Lindameldung(1) an Master geschickt :\n");
  fflush(stdout);
  system("date");
```

```
/****************************************************************************/
/* Senden von Message 1                                                     */
/****************************************************************************/

  while (out(machine[0].name,"MSG","isiiiiii",1,myname,small,part,anzahl,pp,
            (int)(timer[0]*mips()),(int)(timer[1]*mips())) != 0)
    l_error("Fehler in OUT MSG : ");
  anzahl=0;
  while (in("O.K.","SSS", myname, dstart, dende) != 0)
    l_error("Fehler in IN O.K. : ");
  printf("Von Master OK bekommen. Zeit : \n");
  fflush(stdout);
  system("date");
  printf("VOM MASTER STARTD: %s STOPD: %s \n",dstart,dende);

/****************************************************************************/
/*  Uebertragen der Ergebnisdateien                                         */
/*  Falls dabei kein Fehler auftritt wird die Datei geloescht               */
/****************************************************************************/

  if (rcp(machine[0].name,smalldat,smalldat) != 0)
    l_error("Verschicken der Faktorenmatrix : ");
  else
    unlink(smalldat);
  if (rcp(machine[0].name,pdat,pdat) != 0 )
    l_error("Verschicken der PARTION : ");
  else
    unlink(pdat);
  if (rcp(machine[0].name,ppdat,ppdat) != 0)
    l_error("Verschicken der Partion-Partion : ");
  else
    unlink(ppdat);

/****************************************************************************/
/* Senden von Message 2                                                     */
/****************************************************************************/

  while (out(machine[0].name,"MSG","isiiiiii",2,myname,0,0,0,0,0,0) != 0)
    l_error("OUT von Mesaage 2: ");
  printf("Lindameldung(2) an Master geschickt  : ");
  system("date");
  timer[1] = timer[0] = 0;
  start_timing(timer);
}
```

sieben.c

```
#include "hmpqs.h"

extern int SIEBSTART;

#ifdef TEST
            extern SIEBTYP *sie;
#endif

/****************************************************************************/
/* DOC                            SIEBEN()                                  */
/*                                                                          */
/*      Die Funktion fuehrt den Siebvorgang fuer alle Elemente der FB, die  */
/*      groesser sind als FB[siebstart] durch.                              */
/*                                                                          */
/*  zuletzt geaendert am 24.10.92                                           */
/****************************************************************************/
```

```
int sieben(FB, LOGP, START1, START2 ,Testwert, sieb, CANDIDATE, siebint, ende)
     int      *FB, *START1, *START2, *CANDIDATE, siebint;
     SIEBTYP  *LOGP, Testwert, *sieb, *ende;
{
  register int  p, i, l, x, zaehler = 0, *fbp;

/****************************************************************************/
/* Teile das gesamte Siebintervall [-SIEBSTART, SIEBSTART] in Teilintervalle */
/* der Groesse siebint auf. (Diese koennen im Hauptspeicher gehalten werden)  */
/****************************************************************************/

  for (i=-SIEBSTART; i< SIEBSTART;i+= siebint)
    {
      memset(sieb, 0, siebint * sizeof(SIEBTYP));
      l = kleinstart;
      fbp = &FB[kleinstart];
      while ((p = *fbp++) != 0)
        {
          if (START1[l] == START2[l])
            START1[l] = START2[l] = pass(sieb + START1[l], ende, LOGP[l], p);
          else
            {
              START1[l] =  pass(sieb + START1[l], ende, LOGP[l], p);
              START2[l] =  pass(sieb + START2[l], ende, LOGP[l], p);
            }
          l++;
        }

/****************************************************************************/
/* Durchsuche jedes Teilintervall nach Stellen, deren Wert einen Testwert     */
/* uebersteigt. Diese werden im naechsten Modul genauer untersucht.           */
/****************************************************************************/

      for (x=0; x < siebint ; x++)
        if (sieb[x] > Testwert)
          {
            CANDIDATE[zaehler++] = x + i;
#ifdef TEST
           sie[zaehler] = sieb[x];
#endif
        }
    }
  CANDIDATE[zaehler] = 0;
  return(0);
}

/****************************************************************************/
/* FUNKTION: pass(begin, end, logp, p)                                       */
/*                                                                           */
/*      Die Funktion fuehrt einen Siebvorgang in einem Teilintervall mit     */
/*      einer Nullstelle von p durch. Dabei wird immer logp aufaddiert.      */
/*                                                                           */
/*      Eingabe:  int *begin       =  Pointer auf erste Nullstelle           */
/*                int *ende        =  Pointer auf Intervallende              */
/*                int p            =  Primzahl                               */
/*                int logp         =  "dualer Logarithmus" von p             */
/*                                                                           */
/*      Return :  Startwert fuer das naechste Teilintervall                  */
/*                                                                           */
/*      zuletzt geaendert am 29.02.92                                        */
/****************************************************************************/

int pass(begin, end, logp, p)
       SIEBTYP           *begin, *end;
       register int      p;
       register SIEBTYP  logp;
```

```
{
  for(;;)
    {
      if (begin <= end)
        {
          (*begin) += logp;
          begin += p;
        }
      else
        return((begin - end) -1);
    }
}
```

pro_div.c

```
#include "hmpqs.h"

extern double     d_wurz;
extern short      small, part, pp;
extern char       smalldat[],pdat[],ppdat[];
extern LONGTYPE   kN, MINLP, LP, DOUBLE;
extern int        SIEBSTART;

#ifdef TEST
            extern SIEBTYP *sie;
#endif

/*****************************************************************************/
/* DOC                        TESTE()                                        */
/*      Die Funktion untersucht an geeigneten Stellen des Siebarrays, ob ein */
/*      Funktionswert zu einer full, partial oder partial-partial realtion   */
/*      fuehrt, und schreibt im Erfolgsfall die Primfaktorzerlegung der Q(x) */
/*      in die jeweiligen Dateien  FAKTORENMATRIX,PARTION bzw PART.LARGE     */
/*                                                                           */
/*  zuletzt geaendert am 10.06.92                                            */
/*****************************************************************************/

teste(A, B, D2inv, CANDIDATE, FB, START1, START2, faktor)
     int       *FB, *START1, *START2, *CANDIDATE;
     LONGTYPE  A, B, D2inv;
     char      *faktor;
{
  int             rest, fak_i, p1, p2;
  LONGTYPE        H, Q_von_x, erg;
  register int    x, p, vorber, teilt = 0, N1, N2, versuche = 0;
  FILE            *fp, *fpla, *fppl;
  double          a, b;
  char            Hstring[150];

/*****************************************************************************/
/* Allokieren von Speicher fuer die verwendeten langen Zahlen                */
/*****************************************************************************/

  H       = h_get_temp(stel(kN) + 2);
  Q_von_x = h_get_temp(2 *stel(kN) + 2);
  erg     = h_get_temp(stel(kN) + 2);

/*****************************************************************************/
/* Oeffnen der Ergebnisdateien                                               */
/*****************************************************************************/

 if (!(fp = fopen(smalldat,"a")))
   return(-2202);
```

```
 if (!(fpla = fopen(pdat,"a")))
   return(-2203);
 if (!(fppl = fopen(ppdat,"a")))
   return(-2204);

/*****************************************************************************/
/* Berechnen der Nullstellen des Polynoms                                    */
/*****************************************************************************/

      a  = 2 * ldouble(A);
      b  = ldouble(B);
      N1 = (-b - d_wurz) / a;
      N2 = (-b + d_wurz) / a;

  while((x = CANDIDATE[versuche++]) != 0)
    {
      lsmul(A, x << 1, Q_von_x);
      lladd(Q_von_x, B, Q_von_x);
      llmul(Q_von_x, D2inv, Q_von_x);
      lldiv(Q_von_x, kN, 0, H);
      lsquare(H, Q_von_x);
      lldiv(Q_von_x, kN, 0, Q_von_x);

/*****************************************************************************/
/* TESTAUSGABEN                                                              */
/*****************************************************************************/

#ifdef TEST
             printf("Siebwert: %d X: %d "sie[versuche-1],x);
#endif

/*****************************************************************************/
/* Loesche String zur Abspeicherung der Primfaktorzerlegung ueber der FB     */
/*****************************************************************************/

      memset(faktor, 0, 500);

/*****************************************************************************/
/* Ermittele possitive Restdarstellung                                       */
/*****************************************************************************/

      if (N1 < x && x < N2)
        {
          llsub(kN, Q_von_x, Q_von_x);
          sprintf(faktor," 1 1");
        }
      while ((Q_von_x[1] & 1) == 0 )
        {
          teilt++;
          ldiv2(Q_von_x);
        }
      if (teilt)
        {
          sprintf(faktor,"%s %d 2",faktor,teilt);
          teilt = 0;
        }

/*****************************************************************************/
/* Teste fuer alle Faktorbasiselemente < FB[kleinstart], ob sie Q(x) teilen  */
/*****************************************************************************/

      fak_i = 2;
      while (fak_i++ < kleinstart - 1)
        {
          p = FB[fak_i];
          vorber = (SIEBSTART + x) % p;
```

```
          if ((vorber == START1[fak_i]) || (vorber == START2[fak_i]))
            {
              do
                {
                  lsdiv(Q_von_x, p, erg, &rest);
                  if (rest == 0)
                    {
                      teilt++;
                      sto(erg, Q_von_x);
                    }
                } while (rest == 0);
              sprintf(faktor,"%s %d %d",faktor,teilt,fak_i);
              teilt = 0;
            }
        }

/*************************************************************************/
/* Teste fuer die uebrigen Faktorbasiselemente, ob sie Q(x) teilen       */
/*************************************************************************/

      while ((p=FB[fak_i]) != 0)
        {
          vorber = p - (SIEBSTART - x) % p;
          if ( vorber == p )
             vorber = 0;
          if ((vorber == START1[fak_i]) || (vorber == START2[fak_i]))
            {
              do
                {
                  lsdiv(Q_von_x, p, erg, &rest);
                  if (rest == 0)
                    {
                      teilt++;
                      sto(erg, Q_von_x);
                    }
                } while (rest == 0);
              sprintf(faktor,"%s %d %d",faktor,teilt,fak_i);
              teilt = 0;
            }
          fak_i++;
        }

/*************************************************************************/
/* Teste ob Q(x) vollstaendig ueber der Faktorbasis zerfaellt            */
/*************************************************************************/

      if (test1(Q_von_x))
        {

/*************************************************************************/
/* TESTAUSGABEN                                                          */
/*************************************************************************/

          #ifdef TEST
                        printf("FULL ");
          #endif

          lstring(H, Hstring);
          small++;
          fprintf(fp, "%s :%s 0\n", Hstring,faktor);
        }
      else
        {
/*************************************************************************/
/* Partial Relation gefunden ?                                           */
/*************************************************************************/
```

```
      if ((!(lssto(Q_von_x,&rest))) && (rest <LARGEP))
            {

/*****************************************************************************/
/* TESTAUSGABEN                                                              */
/*****************************************************************************/

          #ifdef TEST
                        printf("PARTIAL ");
          #endif

              part++;
              fprintf(fpla,"%9d @ ", rest);
              lstring(H, Hstring);
              fprintf(fpla, " %s :%s 0\n", Hstring,faktor);
            }
          else
            {

/*****************************************************************************/
/* Partial-partial Relation gefunden ?                                       */
/* Ist der Rest zwischen pmax^2 und 10^16, ist es eine Primzahl?             */
/*****************************************************************************/

             if ((llcomp(DOUBLE,Q_von_x) == 1) && (llcomp(Q_von_x,MINLP) == 1))
               {

/*****************************************************************************/
/* Zerlege die Zahl mit Squfof von D. Shanks                                 */
/*****************************************************************************/

                 if ( (!listprim(Q_von_x,2)) && ((p = squfof(Q_von_x, LP,&p1,
                        &p2)) == 2))
                   {

/*****************************************************************************/
/* TESTAUSGABEN                                                              */
/*****************************************************************************/

                      #ifdef TEST
                                  printf("LARGE ");
                      #endif

                      pp++;
                      fprintf(fppl,"%9d %9d @ ",min(p1,p2),max(p1,p2));
                      lstring(H, Hstring);
                      fprintf(fppl, "%s :%s 0\n", Hstring,faktor);
                   }
                  else

/*****************************************************************************/
/* Falls Zerlegung durch Squfof nicht moeglich war probiere es mit Pollardrho */
/*****************************************************************************/

                   if ((p == 1) && (pollard(Q_von_x, LP,&p1,&p2) == 2))
                     {

/*****************************************************************************/
/* TESTAUSGABEN                                                              */
/*****************************************************************************/

                        #ifdef TEST
                                    printf("LARGE ");
                        #endif
```

```
                        pp++;
                        fprintf(fppl,"%9d %9d @ ",min(p1,p2),max(p1,p2));
                        lstring(H, Hstring);
                        fprintf(fppl, "%s :%s 0\n", Hstring,faktor);
                      }
                 }
            }

/*****************************************************************************/
/* TESTAUSGABEN                                                              */
/*****************************************************************************/

         #ifdef TEST
            printf("\n");
         #endif
        }

/*****************************************************************************/
/* TESTAUSGABEN                                                              */
/*****************************************************************************/

  #ifdef TEST
      printf("Versuche : %d FULL : %d PARTIAL : %d PP : %d\n",versuche,small
              ,part,pp);
  #endif

  fclose(fp);
  fclose(fpla);
  fclose(fppl);
  h_mfree_temp(erg,3);
}
```

gen_D.c

```
#include "hmpqs.h"
extern LONGTYPE kN;

/*****************************************************************************/
/* DOC                        GENERIERE_D()                                  */
/*                                                                           */
/*      Die Funktion berechnet das naechste verwertbare D.                   */
/*                                                                           */
/*      Eingabe:  LONGTYPE  D  = letztes D                                   */
/*                                                                           */
/*      Ausgabe:  LONGTYPE  D  = neues D                                     */
/*                                                                           */
/*  zuletzt geaendert am 06.10.92                                            */
/*****************************************************************************/

generiere_D(D)
     LONGTYPE  D;
{
  LONGTYPE temp;

  temp = h_get_temp(stel(kN));

  while (1)
    {
      lsadd(D, 4, D);
      sto(D,temp);
      ldiv2(temp);
      potenz_mod(kN, temp, D, temp);
```

```
      if (!test1(temp))
        continue;
      if (fermat(D) == 1)
        break;
    }
  h_free_temp(temp);
}
```

ber_kor.c

```
#include "hmpqs.h"

extern LONGTYPE  kN;
extern int       SIEBSTART;

/******************************************************************************/
/* DOC              BERECHNE_KOEFFIZIENTEN()                                 */
/*                                                                            */
/*    Die Funktion berechnet aus einem geg. D die Koeffizienten des Polynoms  */
/*    Q(x) und Startwerte zum Sieben                                          */
/*                                                                            */
/*    Eingabe:  int *FB                   = Faktorbais                        */
/*              LONGTYPE D                = zu benutzendes D                  */
/*              int *SQRTkN               = Ressolwurzeln                     */
/*                                                                            */
/*    Ausgabe:  LONGTYPE A, B             = Polynomkoeffizienten              */
/*              LONGTYPE D2inv            = (2D)^-1                           */
/*              int      *START1,*START2  = Startwerte fuer die Siebphase     */
/*                                                                            */
/*    Return :  1     D nicht brauchbar                                       */
/*              -401  Fehler bei Inversenberechnung                           */
/*                                                                            */
/*  zuletzt geaendert am 09.03.92                                             */
/******************************************************************************/

int ber_koeffizienten(D, A, B, D2inv,FB, SQRTkN, START1, START2)
     int       *FB, *SQRTkN, *START1, *START2;
     LONGTYPE  D, A, B, D2inv;
{
  register int  hilf, hilf1, i, p, Bp, Ap, Bph, groesse = FB[0], SIEBS;
  LONGTYPE      temp1, temp2;

  temp1 = h_get_temp(DEFAULT_LENGTH);
  temp2 = h_get_temp(DEFAULT_LENGTH);

/******************************************************************************/
/* Berechne (2D)^-1 mod kN  fuer spaeter                                      */
/******************************************************************************/

  sto(D, temp1);
  lmul2(temp1);
  if (linvers(temp1, kN, D2inv) < 0)
    return(-401);

/******************************************************************************/
/* Berechne h0 : (kN)^(D-3)/4  mod D (in temp2)                              */
/******************************************************************************/

  lsadd(D, -3, temp1);
  ldiv2(temp1); ldiv2(temp1);
  potenz_mod(kN, temp1, D, temp2);
```

```
/*****************************************************************************/
/*  Berechne h1 : kN * ho :=(kN)^(D+1)/4  mod D (in temp1)                   */
/*****************************************************************************/

  llmul(kN,temp2,temp1);
  lldiv(temp1, D, 0, temp1);
  sto(temp1,B);

/*****************************************************************************/
/* Berechne (2*h1)^(-1) mod D (in temp2)                                     */
/*****************************************************************************/

  lsadd(D,1,A);
  ldiv2(A);
  llmul(A,temp2,temp2);
  lldiv(temp2, D, 0, temp2);

/*****************************************************************************/
/* Berechne h2 : (2*h1)^(-1)((kN - h1^2)/D) mod D (in temp1)                 */
/*****************************************************************************/

  lsquare(temp1,temp1);
  if (llcomp(kN, temp1) >= 0)
    llsub(kN, temp1, temp1);
  else
    llsub(temp1, kN, temp1);
  lldiv(temp1, D, temp1, A);
  if (!test0(A))
    {
      h_mfree_temp(temp2,2);
      return(1);
    }
  llmul(temp1, temp2, temp1);
  lldiv(temp1, D, 0, temp1);

/*****************************************************************************/
/* Berechne B = h2 * D + h1 mod A , A = D^2                                  */
/*****************************************************************************/

  llmul(temp1, D, A);
  lladd(A, B, B);
  lsquare(D, A);
  lldiv(B, A, 0, B);
  if (lmod2(B) == 0)
    llsub(A, B, B);

/*****************************************************************************/
/* Ausgabe zu Testzwecken                                                    */
/*****************************************************************************/

#ifdef TEST
              printf("A = ");
              lwrite(A);
              printf("B = ");
              lwrite(B);
              lsquare(B,temp1);
              llsub(kN,temp1,temp2);
              ldiv2(temp2);
              ldiv2(temp2);
              lldiv(temp2,A,temp2,temp1);
              printf("C = ");
              lwrite(temp2);
              if (!test0(temp1))
              printf("Fehler bei der Berechnung der Koeffizienten \n");
              fflush(stdout);
#endif
```

```
/*****************************************************************************/
/* Berechnung der Startwerte fuer die Siebphase fuer alle Faktorbasiselemente */
/*****************************************************************************/

  for (i = 3; i <= groesse + 1; i++)
    {
      p = FB[i];
      SIEBS = SIEBSTART % p;
      Bph = Bp = lsmod(B, p);
      if (Ap = lsmod(A, p))
        {
          if ((Bp = SQRTkN[i] - Bp) < 0)
            Bp += p;
          Ap = sinvers((Ap<<1), p);
          if (p <WURZINT)
            hilf = (Ap * Bp ) % p;
          else
            {
              slsto(Ap, temp1);
              lsmul(temp1, Bp, temp2);
              hilf = lsmod(temp2, p);
            }
          if ((hilf1 = SIEBS + hilf) >= p )
            START1[i] = hilf1 - p ;
          else
            START1[i] = hilf1;

          if ((Bp = (-SQRTkN[i] - Bph) % p) < 0)
            Bp += p;
          if (p <WURZINT)
             hilf = (Ap * Bp ) % p;
          else
            {
              lsmul(temp1, Bp, temp2);
              hilf = lsmod(temp2, p);
            }
          if ((hilf1 = SIEBS + hilf) >= p)
            START2[i] = hilf1 - p ;
          else
            START2[i] = hilf1;
        }
      else
        {
          lsquare(B, temp1);
          if (llcomp(kN, temp1) >= 0)
            llsub(kN, temp1, temp1);
          else
            {
              llsub(temp1, kN, temp1);
              Bp = p - Bp;
            }
          ldiv2(temp1); ldiv2(temp1);
          lldiv(temp1, A, temp1, 0);
          Bp = sinvers(Bp, p);
          lsmul(temp1, Bp, temp1);
          hilf = lsmod(temp1, p);
          if ((hilf1 = SIEBS + hilf) >= p )
            START1[i] = START2[i] = hilf1 - p ;
          else
            START1[i] = START2[i] = hilf1;
        }
    }
  h_mfree_temp(temp2,2);
  return(0);
}
```

A.1.3 Die single Version

Das Makefile

```
################################################################################
#                                                                              #
#                          Makefile fuer PMPQS-SINGLE                          #
#                                                                              #
################################################################################

VER=4
MACHINE=-sun$(VER)

INCLUDE=/home/denny/INCLUDE
LIBDIR=/home/denny/lib

LIBS=$(LIBDIR)/sieve.a /home/berger/DIPLOM/LARITH/larith.a  /usr/lib/libm.a

OBJ=  pmpqs.o faktor_N.o sieben.o pro_div.o gen_D.o ber_kor.o

DEBUG=

OPT= -O4
.KEEP_STATE:

tpmpqs: $(LIBS) $(OBJ)
cc $(OPT) $(MACHINE) -o tpmpqs $(OBJ) $(LIBS)

pmpqs.o: pmpqs.c
cc $(OPT) $(MACHINE) $(DEBUG) -I$(INCLUDE) -c pmpqs.c

faktor_N.o : faktor_N.c
cc $(OPT) $(MACHINE) $(DEBUG) -I$(INCLUDE) -c faktor_N.c

sieben.o: sieben.c
cc $(OPT) $(MACHINE) $(DEBUG) -I$(INCLUDE) -c sieben.c

pro_div.o: pro_div.c
cc $(OPT) $(MACHINE) $(DEBUG) -I$(INCLUDE) -c pro_div.c

gen_D.o: gen_D.c
cc $(OPT) $(MACHINE) $(DEBUG) -I$(INCLUDE) -c gen_D.c

ber_kor.o: ber_kor.c
cc $(OPT) $(MACHINE) $(DEBUG) -I$(INCLUDE) -c ber_kor.c
```

faktor_N.c

```
#include "hmpqs.h"

short     small = 0 ,part = 0, pp = 0;
int       SIEBSTART;
char      smalldat[80], pdat[80], ppdat[80];
LONGTYPE  MINLP,DOUBLE,LP,kN;
double    d_wurz;

#ifdef TEST
            SIEBTYP *sie;
#endif
```

```
/****************************************************************************/
/* DOC                          FAKTORISIERE_N()                            */
/*                                                                          */
/*      Die Funktion versucht die Zahl N mit dem QS-Sieb-Algorithmus zu     */
/*      faktorisieren. Sie ruft die dazu notwendigen Module auf.            */
/*                                                                          */
/*      Eingabe:  LONGTYPE N           = zu fakt. Zahl                      */
/*                                                                          */
/*      Ausgabe:  LONGTYPE *teiler?    = Teiler der Zahl N                  */
/*                                                                          */
/*      Return :  0                    = ohne Fehler                        */
/*                fehlernr             = sonst                              */
/*                                                                          */
/*      zuletzt geaendert am 06.07.92                                       */
/****************************************************************************/
int faktorisiere_N(N,teiler1, teiler2)
     LONGTYPE  N, teiler1, teiler2;
{
  FILE          *fp;
  LONGTYPE      D, A, B, D2inv;
  register int  k, p, i, rcode, testwert;
  int           groesse_FB, M, *SQRTkN = NULL, *FB = NULL, *START1 = NULL;
  int           *START2 = NULL, *CANDIDATE = NULL, stellenzahl;
  SIEBTYP       *sieb = NULL, *ende, *LOGP = NULL;
  char          zeile[100], *faktor,faktorisiert = 0;
  double        T;

  kN = lalloc(stel(N) + 5);

/****************************************************************************/
/*   Bestimmung des Multiplier k                                            */
/*   Die Knuth-Schroeppelfunktion wird fuer BIS_PRIMZAHL viele Faktorbasis- */
/*   elemente ausgewertet                                                   */
/****************************************************************************/

  if (( k = finde_k(N, BIS_PRIMZAHL)) < 0)
    return(k);
  printf("Multiplier k = %d\n", k);

  lsmul(N, k, kN);
  lstring(kN,zeile);
  stellenzahl = strlen(zeile);

/****************************************************************************/
/*   Speicherallokieren fuer Parameter der Polynomerzeugung D und eine      */
/*   Hilfsvariable                                                          */
/****************************************************************************/

  DOUBLE    = lalloc(6);
  MINLP     = lalloc(6);
  LP        = lalloc(6);
  D         = lalloc(stel(kN)+1);
  A         = lalloc(2 * stel(kN));
  B         = lalloc(2 * stel(kN));
  D2inv     = lalloc(2 * stel(kN));

/****************************************************************************/
/*   Einlesen der Parameter ( Toleranzwert, Siebintervallaenge, Groesse der */
/*   Faktorbasis ) fuer eine Zahl mit stellenanzahl Dezimalstellen          */
/****************************************************************************/
```

```
  if ((rcode = get_parameter(stellenzahl, &T, &M, &groesse_FB)) < 0)
    return(rcode);

  printf("Toleranz: %05.2lf\nFB: %d\nSiebgroesse : %d\n",T, groesse_FB,M);
  printf("Index der 1.Siebprimzahl:%d\n\n",kleinstart);
  fflush(stdout);
  SIEBSTART = M;

#ifdef TEST
    anf_zeit(0);
#endif

/*************************************************************************/
/*   Erstellen der Faktorbasis fuer kN                                   */
/*************************************************************************/

  if ((rcode = bestimme_FB(groesse_FB, kN, &FB,  k)) < 0)
    return(rcode);

/*************************************************************************/
/*   Verlasse die Prozedur falls ein Teiler beim Erstellen der Faktorbasis   */
/*   gefunden wurde.                                                     */
/*************************************************************************/

  if (rcode > 1)
    {
      slsto(rcode, teiler1);
      llsto(teiler1, teiler2);
      return(0);
    }

/*************************************************************************/
/*   Berechnung des Testwertes mit dem jeder Siebeintrag nach dem Sieben     */
/*   verglichen wird.                                                    */
/*************************************************************************/

  testwert = LOGMUL * (SIEBTYP)(0.5 * log2(ldouble(kN)) + log2((double)M) -
             T*log2((double)FB[FB[0]+1]));

  printf("TESTWERT : %d\n",testwert);

/*************************************************************************/
/*   Berechnung der Ressolwurzeln und Allokieren des benoetigten Speicherplatz*/
/*   Die Werte werden in einem Integer Feld SQRTkN abgespeichert und zum Ver- */
/*   schicken an die Clienten in die Datei RESSOLW abgespeichert.        */
/*************************************************************************/

  if (!(sieb = (SIEBTYP *)malloc(SIEBINTERN * sizeof(SIEBTYP))))
    return(-1105);
  ende = sieb + SIEBINTERN - 1;

  if (!(faktor = (char *)malloc(500 * sizeof(char))))
    return(-1107);

  if (!(LOGP = (SIEBTYP *)calloc (groesse_FB + 3 , sizeof(SIEBTYP))))
    return(-1111);

  if (!(SQRTkN = (int *)calloc (groesse_FB + 3 , sizeof(int))))
    return(-1110);
```

```
  if (!(START1 = (int *)calloc (groesse_FB + 3 , sizeof(int))))
    return(-1112);
  if (!(START2 = (int *)calloc (groesse_FB + 3 , sizeof(int))))
    return(-1117);

  if (!(CANDIDATE = (int *)calloc(600,sizeof(int))))
    return(-1106);

#ifdef TEST
  if (!(sie = (SIEBTYP *)calloc(600,sizeof(SIEBTYP))))
    return(-1106);
#endif

  for (i = 2; i <= groesse_FB + 1; i++)
    {
      p = FB[i];
      LOGP[i] = (SIEBTYP)(log2((double)p)*LOGMUL);
      if ((SQRTkN[i] = m_ressol(lsmod(kN, p), p)) < 0)
        SQRTkN[i] += p;
    }

  if (!(fp=fopen("RESSOLW","w")))
    return(-2323);
  for (i=2; i <= groesse_FB + 2 ;i++)
    fprintf(fp,"%d\n",SQRTkN[i]);
  fclose(fp);

  d_wurz = sqrt(ldouble(kN));

  slsto(DOUBLEP,DOUBLE);
  slsto(LARGEP,LP);
  lsquare(DOUBLE,DOUBLE);
  slsto(FB[groesse_FB+1],MINLP);
  lsquare(MINLP,MINLP);
  printf("LARGE : ");
  lwrite(LP);
  printf("DOUBLE : ");
  lwrite(DOUBLE);
  fflush(stdout);

/******************************************************************************/
/*   Test ob das Programm aufgesetzt wurde ( dann existiert eine Datei LAST_D */
/*   mit dem letzten verschickten Wert fuer D). Sonst wird der Wert fuer      */
/*   STARTD ( = ((Sqrt(kN/2)/M) berechnet. Ausserdem wird sichergestellt, dass*/
/*   STARTD kongruent 3 modulo 4 ist.                                         */
/******************************************************************************/

  if (fp=fopen("LAST_D","r"))
    {
      flread2(fp,D);
      fclose(fp);
      printf("Program wurde wiederaufgesetzt. Last_D = ");
      lwrite(D);
      fflush(stdout);
    }
  else
    {
      lsdiv(kN, 2, A, NULL);
      longsqrt(A, A);
```

```
      lsdiv(A, M, A, NULL);
      longsqrt(A, D);
      if (!lscomp(D, 0))
        slsto(3, D);
      D[1] = D[1] | 3;
      lsadd(D,-4,D);
    }

/*****************************************************************************/
/*  Bereitstellen der Dateien zur Speicherung der Ergebnisse                 */
/*  fuer full relations              FAKTORENMATRIX                          */
/*  fuer partial relations           PARTION                                 */
/*  fuer partial-partial relations   PART.LARGE                              */
/*****************************************************************************/

  sprintf(smalldat,"%s","FAKTORENMATRIX");
  sprintf(pdat,"%s","PARTION");
  sprintf(ppdat,"%s","PART.LARGE");

#ifdef TEST
            end_zeit("bestimme_FB & co");
            anf_zeit(1);
#endif

while (1)
    {
      small = part = pp = 0;
      do {
           generiere_D(D);
           if ((rcode = ber_koeffizienten(D, A, B, D2inv, FB, SQRTkN, START1,
                                          START2) < 0))
              return(rcode);
         } while(rcode != 0);

#ifdef TEST
            end_zeit("berechne_ko");
            anf_zeit(1);
#endif

      if ((rcode = sieben(FB, LOGP, START1, START2,testwert,sieb, CANDIDATE,
                          SIEBINTERN, ende) < 0))
        return(rcode);

#ifdef TEST
            end_zeit("sieben");
            anf_zeit(1);
#endif

      if ((rcode = teste(A, B, D2inv, CANDIDATE, FB, START1, START2, faktor))
            < 0)
        return(rcode);

#ifdef TEST
            end_zeit("teste");
            anf_zeit(1);
#endif

     }
  return(rcode);
}
```

A.1.4 Verwalten der Relationen

Das Makefile

```
################################################################################
#                                                                              #
#                             Makefile fuer ZUSAM                              #
#                                                                              #
################################################################################

VER=4
MACHINE=-sun$(VER)

INCLUDE=/home/denny/fopra/include
LIBDIR=/home/denny/lib
LIBS=$(LIBDIR)/sieve.a /home/berger/DIPLOM/LARITH/larith.a /usr/lib/libm.a

OBJ= zusam.o

DEBUG=

OPT= -O4

zusam: $(LIBS) $(OBJ)
cc $(OPT) $(MACHINE) -o zusam $(OBJ) $(LIBS)

zusam.o: zusam.c
cc $(OPT) $(MACHINE) $(DEBUG) -I$(INCLUDE) -c zusam.c
```

zusam.c

```
#include "hmpqs.h"

#define anzahl 40
#define zeilen 40000
#define S 1200000

LONGTYPE   hilf1, X_quad, Y_quad, kN, FBh , hilf;
int        *DAT, *VERWEIS, *FB, groesse_FB, REF[anzahl], NEUREF[anzahl];
FILE     *fp, *fp1, *fpin;
char       hilfe[2000], zeile[2000], fzeile[2000];
int        eof,c = 0 ,v = 0  ,e = 0, p1, p2, small, parti,alt,aktuell;

/******************************************************************************/
/* DOC                         ZUSAM                                          */
/*                                                                            */
/*      Das Programm fuegt neue Relationen in die bestehenden Dateien ein     */
/*      Es testet ob die Relationen korrekt sind und zaehlt die Anzahl der    */
/*      Relationen, die bisher gefunden sind, und ermittelt die Gesamtzahl    */
/*      der full realtions (normale + kombinierte).                           */
/*                                                                            */
/*      zuletzt geaendert am 06.03.93                                         */
/******************************************************************************/

main()
{
  char          h1_string[150];
  register int  i=0;
  int           stell;
```

```
/*****************************************************************************/
/  Starten der Langen Arithmetik                                              */
/*****************************************************************************/

  linit(DEFAULT_LENGTH);

/*****************************************************************************/
/*  Einlesen der Faktorbasis und der zu faktorisierenden Zahl aus der Datei   */
/*  FAKTORBASIS                                                               */
/*****************************************************************************/

  if (!(fp  = fopen("FAKTORBASIS","r")))
    fehler(-99998);

  fscanf(fp, "%d", &groesse_FB);
  fscanf(fp, "%s", zeile);
  stell = (int)(2.302585 * strlen(zeile)/32) + 5;
  kN = lalloc(stell);
  string_to_l(zeile,kN);

  if (FB)
    free(FB);
  if (!(FB = (int *)calloc (groesse_FB + 3 , sizeof(int))))
    fehler(-3456);

  FB[0] = groesse_FB;

  for (i=1; i <= groesse_FB + 2 ;i++)
    fscanf(fp,"%d",&FB[i]);
  fclose(fp);

/*****************************************************************************/
/* Allokieren von Speicher fuer lange Zahlen                                  */
/*****************************************************************************/

  X_quad   = lalloc(2 * stel(kN) + 4);
  Y_quad   = lalloc(2 * stel(kN) + 4);
  hilf     = lalloc(2 * stel(kN) + 4);
  hilf1    = lalloc(2 * stel(kN) + 4);
  FBh      = lalloc(5);

/*****************************************************************************/
/* Sind neue full relations da?                                               */
/* Falls ja lese sie ein, teste ob sie fehlerfrei sind und sortiere in die    */
/* bestehende Datei SMALL ein.                                                */
/*****************************************************************************/

  if (fcheck("FAKTORENMATRIX"))
    {
      fp  = fopen("FEHLER.SMALL","a");
      fp1 = fopen("OK.SMALL","w");
      fpin = fopen("FAKTORENMATRIX","r");
      while (fgets(zeile, 2000, fpin) != 0)
        {
          sscanf(zeile, "%s",h1_string);
          reltest(1,1,h1_string);
         }
      fclose(fp);
      fclose(fp1);
      fclose(fpin);
      if (!fcheck("SMALL"))
        system("sort -u OK.SMALL > SORT.SMALL");
      else
        {
          system("sort -u OK.SMALL > SMALL.sort; sort -u -m SMALL.sort SMALL >
                  SORT.SMALL");
```

```
          unlink("SMALL.sort");
          unlink("SMALL");
         }
      unlink("OK.SMALL");
      system("mv SORT.SMALL SMALL");
    }
/****************************************************************************/
/* Ermittele die Anzahl der gefundenen full relations                       */
/****************************************************************************/

  small = zaehle("SMALL");

/****************************************************************************/
/* Sind neue partial relations da?                                          */
/* Falls ja lese sie ein, teste ob sie fehlerfrei sind und sortiere in die  */
/* bestehende Datei PARTION ein.                                            */
/****************************************************************************/

  if (fcheck(get_name("partion")))
    {
      fp  = fopen("FEHLER.PART","a");
      fp1 = fopen("OK.PART","w");
      fpin = fopen(get_name("partion"),"r");
      while (fgets(zeile, 2000, fpin) != 0)
        {
          sscanf(zeile, "%d @ %s",&p1, h1_string);
          reltest(1,p1,h1_string);
        }
      fclose(fp);
      fclose(fp1);
      fclose(fpin);
/****************************************************************************/
/* Sortiere die neuen korrekten partial-Relations in die Bestehenden Dateien */
/* ein. Teste dann, ob die neuen Dateien, eine bestimmte Groesse ueberschrei- */
/* ten. Falls ja werden sie geteilt um leichter Handhabar zu sein           */
/****************************************************************************/

      einsort("OK.PART","PART");
      kuerze("PART");
    }

/****************************************************************************/
/* Sind neue partial-partial relations da?                                  */
/* Falls ja lese sie ein, teste ob sie fehlerfrei sind und sortiere in die  */
/* bestehende Datei PART.LARGE ein.                                         */
/****************************************************************************/

  if (fcheck(get_name("ppartion")))
    {
      fp  = fopen("FEHLER.LARGE","a");
      fp1 = fopen("OK.LARGE","w");
      fpin = fopen(get_name("ppartion"),"r");
      while (fgets(zeile, 2000, fpin) != 0)
        {
          sscanf(zeile, "%d %d @ %s",&p1, &p2, h1_string);
          reltest(p1,p2,h1_string);
        }
      fclose(fp);
      fclose(fp1);
      fclose(fpin);
/****************************************************************************/
/* Einsortieren und Sicherstellen der Handhabbarkeit                        */
/****************************************************************************/

      einsort("OK.LARGE","DOUBLE");
      kuerze("DOUBLE");
```

```
    }
/*****************************************************************************/
/* Zaehle die Anzahl der Zykel (full relations) , die durch die gesammelten   */
/* partial und partial-partial Relationen gebildet werden.                    */
/*****************************************************************************/

  zykelzaehl("PART","DOUBLE");
}

/*****************************************************************************/
/* FUNKTION : zykelzaehle(partfile,largefile)                                */
/*                                                                           */
/* Die Funktion zaehlt die Anzahl der Zykel, die durch p und pp Relationen   */
/* gebildet werden. Dazu wird ein Graph aufgebaut, dessen Knoten, Kanten und */
/* verbundenen Komponenten gezaehlt werden. Damit kann die Anzahl der funda- */
/* mentalen Zykel berechnet werden (vergleiche LenMan92)                     */
/*****************************************************************************/

int zykelzaehl (partfile , largefile)
     char partfile[30], largefile[30];
{
  register int i = 0;
  int p1,p2;

  if (!(DAT = (int *)malloc (S * sizeof(int))))
    return(-12);
  if (!(VERWEIS = (int *)malloc (S * sizeof(int))))
    return(-13);
  memset(DAT, -1 ,S * sizeof(int));
  sprintf(zeile,"%s.%d",partfile,i++);
  while (fp = fopen(zeile, "r"))
    {
      while(fgets(zeile,2000,fp))
        {
          e++;
          sscanf(zeile,"%d ",&p2);
          arbeite(1,p2);
        }
      fclose(fp);
      sprintf(zeile,"%s.%d",partfile,i++);
    }
  printf("%5d %6d %6d ",small,e,e+c-v);
  parti = e ;
  i=0;
  sprintf(zeile,"%s.%d",largefile,i++);
  while (fp = fopen(zeile, "r"))
    {
      while(fgets(zeile,2000,fp))
        {
          e++;
          sscanf(zeile,"%d %d",&p1,&p2);
          arbeite(p1,p2);
        }
      fclose(fp);
      sprintf(zeile,"%s.%d",largefile,i++);
    }
  printf("%6d  %6d \n",e-parti,e+c-v);
  fflush(stdout);
}
/*****************************************************************************/
/* FUNKTION : arbeite (p1,p2)                                                */
/*                                                                           */
/* Die Funktion fuegt die durch p1 und p2 repraesentierte Relation in die    */
/* Datenstruktur zur Verwaltung des Graphen ein, und aktualisiert die Anzahl */
/* der Kanten, Knoten und Komponenten                                        */
/*****************************************************************************/
```

```
arbeite(p1,p2)
     int p1, p2;
{
  register int j1,j2, hilf,hilf1, r1, r2;

  j1 = p1 % S;
  j2 = p2 % S;

  while ((DAT[j1] != -1) && (DAT[j1] !=p1))
    j1 = (j1 < S-1)?j1+1:0;
  if (DAT[j1] == -1)
    {
      DAT[j1] = p1;
      VERWEIS[j1] = j1;
      v++;
      c++;
    }

  while ((DAT[j2] != -1) && (DAT[j2] !=p2))
    j2 = (j2 < S-1)?j2+1:0;
  if (DAT[j2] == -1)
    {
      DAT[j2] = p2;
      VERWEIS[j2] = j2;
      v++;
      c++;
    }

  r1 = VERWEIS[j1];
  while ( r1 != VERWEIS[r1])
    r1 = VERWEIS[r1];

  hilf = VERWEIS[j1];
  hilf1 = j1;
  while (hilf != r1)
    {
      VERWEIS[hilf1] = r1;
      hilf1 = hilf;
      hilf = VERWEIS[hilf1];
    }

  r2 = VERWEIS[j2];
  while ( r2 != VERWEIS[r2])
    r2 = VERWEIS[r2];

  hilf = VERWEIS[j2];
  hilf1 = j2;
  while (hilf != r2)
    {
      VERWEIS[hilf1] = r2;
      hilf1 = hilf;
      hilf = VERWEIS[hilf1];
    }

  if (r1 != r2)
    c--;

  if (DAT[r1] < DAT[r2])
    VERWEIS[r2] = r1;
  else
    if(DAT[r2] < DAT[r1])
      VERWEIS[r1] = r2;
}
```

```
/****************************************************************************/
/* FUNKTION : kuerze (was)                                                  */
/*                                                                          */
/* Die Funktion testet, ob die einzelnen Dateien zur Abspeicherung der Rela-  */
/* tionen zuviele Zeilen enthalten. Falls ja werden sie geteilt und die       */
/* Datenstruktur REF wird aktualisiert.                                       */
/****************************************************************************/

kuerze (was)
     char was[30];
{
  register j;
  int vorhanden;

  alt = 0;
  aktuell = 0;
  sprintf(zeile,"REFERENZ.%s",was);
  if (fcheck(zeile))
    {
      fp = fopen(zeile,"r");
      while (fscanf(fp,"%d",&REF[alt++])!= EOF);
      fclose(fp);
      REF[--alt] = 100000000;
    }
  else
    REF[alt++] = 100000000;

  vorhanden = alt;
  alt = 0;
  sprintf(zeile,"%s.0",was);
  if ((j = zaehle(zeile)) > zeilen )
    {
      teile (zeile , (int) (j /2), was);
      unlink(zeile);
      while (alt++ < vorhanden)
        {
          sprintf(zeile,"%s.%d",was,alt);
          if ((j = zaehle(zeile)) > zeilen )
            {
              teile (zeile , (int) (j /2), was);
              unlink(zeile);
            }
          else
            {
              sprintf(zeile,"mv %s.%d hilf.%s.%d",was,alt,was,aktuell);
              system(zeile);
              NEUREF[aktuell++] = REF[alt];
            }
        }
      sprintf(zeile,"REFERENZ.%s",was);
      fp =  fopen(zeile,"w");
      alt = 0;
      while (NEUREF[alt] != 100000000)
        {
          fprintf(fp,"%d\n",NEUREF[alt]);
          sprintf(zeile,"mv hilf.%s.%d %s.%d",was,alt,was,alt);
          system(zeile);
          alt++;
        }
      sprintf(zeile,"mv hilf.%s.%d %s.%d",was,alt,was,alt);
      system(zeile);
    }
}
```

```
/*****************************************************************************/
/* FUNKTION : zaehle (zeile)                                                 */
/*                                                                           */
/* Die Funktion zaehlt die Zeilenanzahl der Datei zeile falls sie existiert   */
/*****************************************************************************/

int zaehle ( zeile)
     char zeile[100];
{
  register int j = 0;

  if (fcheck(zeile))
    {
      fp = fopen (zeile,"r");
      while (fgets(fzeile,2000,fp) != NULL)
        j ++;
    }
  else
    return (0);
  return (j);
}

/*****************************************************************************/
/* FUNKTION : teile (zeile,anz,argu)                                         */
/*                                                                           */
/* Die Funktion teilt die Datei zeile in die Datei hilf.zeile.argu           */
/* argu enthaelt die Nummern der neuen Datei. Es wird sichergestellt, dass   */
/* Relationen mit der selben large prime nicht getrennt werden.              */
/*****************************************************************************/

teile ( zeile, anz, argu)
     int anz;
     char zeile[100], argu[40];
{
  char hilf[100];
  FILE *fpin;
  int vergleich,neu;

  sprintf(fzeile,"hilf.%s.%d",argu,aktuell);
  fp =  fopen(fzeile,"w");

  fpin = fopen (zeile,"r");
  while (fgets(fzeile,2000,fpin) != NULL)
    {
      anz--;
      fprintf(fp,"%s",fzeile);
      if ( anz == 0)
        {
          sscanf(fzeile,"%d",&vergleich);
          fgets(fzeile,2000,fpin);
          sscanf(fzeile,"%d",&neu);
          while (neu == vergleich)
            {
              fprintf(fp,"%s",fzeile);
              fgets(fzeile,2000,fpin);
              sscanf(fzeile,"%d",&neu);
            }
          fclose(fp);
          NEUREF[aktuell++] = vergleich;
          sprintf(hilf,"hilf.%s.%d",argu,aktuell);
          fp = fopen(hilf,"w");
          fprintf(fp,"%s",fzeile);
          while (fgets(fzeile,2000,fpin) != NULL)
            fprintf(fp,"%s",fzeile);
        }
    }
```

```
  NEUREF[aktuell++] = REF[alt];
  fclose(fp);
}

/*****************************************************************************/
/* FUNKTION : einsort (woher,wohin)                                          */
/*                                                                           */
/* Die Funktion sortiert die neuen getesteten Relationen aus der Datei woher */
/* in die richtige Teildatei wohin.*                                         */
/* Die Datei woher ist sortiert und wird mithilfe der REF-werte in Teildateien*/
/* aufgeteilt.                                                               */
/*****************************************************************************/
einsort(woher, wohin )
     char woher[30], wohin[30];
{
  FILE      *fakneu;
  int       vergleich, i = 0;
  char      fzeile[2000];

  sprintf(zeile,"REFERENZ.%s",wohin);

/*****************************************************************************/
/* Falls die Ergebnisdatei bereits aufgespalten wurden,enthaelt die Datei    */
/* REFERENZ.wohin die large-primes, die in der letzten Relation der Teildatei */
/* Abgespeichert ist.                                                        */
/*****************************************************************************/

 if (fcheck(zeile))
    {
      fp = fopen(zeile,"r");
      while (fscanf(fp,"%d",&REF[i++])!= EOF);
      fclose(fp);
      REF[--i] = 100000000;
    }
  else
    REF[0] = 100000000;
  sprintf(zeile,"sort -n %s >%s.sort ",woher,woher);
  system(zeile);
  unlink(woher);
  sprintf(zeile,"%s.sort",woher);
  fpin = fopen(zeile, "r");
  i = 0;
  fgets(fzeile,2000,fpin);
  sscanf(fzeile,"%d",&vergleich);
  if (REF[0] == 100000000)
    {
      sprintf(zeile,"%s.0",wohin);
      if (fcheck(zeile))
        {
          sprintf(zeile,"sort -u -m %s.0 %s.sort > NEU.%s",wohin,woher,wohin);
          system(zeile);
          sprintf(zeile,"%s.0",wohin);
          unlink(zeile);
          sprintf(zeile,"mv NEU.%s %s.0",wohin,wohin);
          system(zeile);
        }
      else
        {
          sprintf(zeile,"mv %s.sort %s.0",woher,wohin);
          system(zeile);
        }
      sprintf(zeile,"%s.sort",woher);
      unlink(zeile);
    }
  else
    {
```

```
      while(REF[i] != 100000000)
        {
          while ( vergleich > REF[i] )
           i++;
          sprintf(zeile,"NEU.%s.%d",woher,i);
          fakneu = fopen(zeile,"w");
          fprintf(fakneu,"%s",fzeile);
          while( eof = (fgets(fzeile,1000,fpin) != NULL))
            {
              sscanf(fzeile,"%d",&vergleich);
              if (vergleich > REF[i])
                break;
              fprintf(fakneu,"%s",fzeile);
            }
          fclose(fakneu);
          if (!eof)
            break;
        }
      sprintf(zeile,"%s.sort",woher);
      unlink(zeile);
      for (i=0;i<anzahl;i++)
        {
          sprintf(zeile,"NEU.%s.%d",woher,i);
          if (fakneu = fopen(zeile,"r"))
            {
              sprintf(zeile,"sort -u -m %s.%d NEU.%s.%d >SORT.%d",wohin,i,woher,
                      i,i);
              system(zeile);
              sprintf(zeile,"NEU.%s.%d",woher,i);
              unlink(zeile);
              sprintf(zeile,"%s.%d",wohin,i);
              unlink(zeile);
              sprintf(zeile,"mv SORT.%d %s.%d",i,wohin,i);
              system(zeile);
            }
          fclose(fakneu);
        }
    }
  fclose(fpin);
}

/*****************************************************************************/
/* FUNKTION : reltest (p1,p2,Hstring)                                        */
/*                                                                           */
/* Die Funktion testet, ob die Relation korrekt ist. Dazu wandelt sie Hstring */
/* in eine lange Zahl um. Dann werden die beiden large primes p1, p2 ( wobei  */
/* jede gleich 1 sein kann ( p1+p2=1 -> full relation ) oder nur p1=1 ->      */
/* partial Relation) und die Primfaktorzerlegung ueber der Faktorbasis (aus   */
/* zeile) aufmultipliziert. Jetzt muessen beide Zahlen kongruent modulo N sein*/
/*****************************************************************************/
reltest(p1, p2, Hstring)
     int      p1, p2;
     char     Hstring[150];
{
  char           *p= NULL, *pp = NULL;
  register int   e, pz;

  strcpy(hilfe,zeile);
  string_to_l(Hstring, Y_quad);
  slsto(p1,hilf);
  slsto(p2,X_quad);
  llmul(X_quad,hilf,X_quad);
  if (pp = strchr(zeile, ':'))
    {
      pp += 2;
      p = strtok(pp, " \n");
```

```
      while (p != NULL)
        {
          e = atoi(p);
          if (!e)
          break;
          p = strtok(NULL, " \n");
          if ((pz = atoi(p)) <groesse_FB + 3)
            {
              slsto(FB[pz], FBh);
              while (e-- != 0)
                 llmul(X_quad, FBh, X_quad);
              lldiv(X_quad, kN, 0, X_quad);
            }
          p = strtok(NULL, " \n");
        }
    }
  lsquare(Y_quad,Y_quad);
  lldiv(Y_quad, kN, 0, Y_quad);
  if (lsig(X_quad) == -1)
    lladd(kN,X_quad,X_quad);

  if ((llcomp(X_quad,Y_quad) != 0) )
    {
      fprintf(fp,"%s",hilfe);
      fflush(fp);
    }
  else
    {
      fprintf(fp1,"%s",hilfe);
      fflush(fp1);
    }
}
```

A.1.5 Konstruieren der quadratischen Kongruenz

wert_aus.c

```
#include "hmpqs.h"
#include <string.h>

/****************************************************************************/
/* DOC                    wert_aus                                          */
/*                                                                          */
/*      Die Funktion berechnet aus den Loesungen des Gleichungssystems echte  */
/*      Teiler von N                                                        */
/*                                                                          */
/*      zuletzt geaendert am 06.10.92                                       */
/****************************************************************************/

main ()
{
  LONGTYPE         N, X_quad, Y_quad, kN, FBh , hilf;
  int              *FB, groesse_FB, loesung = 1, *exponent = NULL;
  FILE             *faktorenmatrix = fopen("FAKTORENMATRIX","r");
  FILE             *allsolutions, *fp;
  register int     zaehler = 0, e, j, i, eoftest;
  int              gefunden = 0, stell ,zeilenindex = 0, rest;
  char             *p= NULL, *pp = NULL, zeile[2000], Hstring[2000];
  long             cand[]= { 1,3,5,7,11,13,15,17,19,21,23,29,31,33,35,37,39,41,
                            43,47,51,53,55,57,59,61,65,67,71,73,79,83,85,89,91,
                            93,95,97,101,103,105,107,109,113,127,131,137,139};

/****************************************************************************/
/*  Initialisieren der lange Arithmetik                                     */
/****************************************************************************/

  linit(DEFAULT_LENGTH);

/****************************************************************************/
/* Einlesen der Faktorbasis und der zu faktorisierenden Zahl                */
/****************************************************************************/

if (!(fp  = fopen("FAKTORBASIS","r")))
    return(-999998);

  fscanf(fp, "%d", &groesse_FB);
  fscanf(fp, "%s", zeile);

  if (!(FB = (int *)calloc (groesse_FB + 3 , sizeof(int))))
    fehler(-3456);

  FB[0] = groesse_FB;

  for (i=1; i <= groesse_FB + 2 ;i++)
    fscanf(fp,"%d",&FB[i]);
  fclose(fp);

  stell = (int)(2.302585 * strlen(zeile)/32) + 5;

/****************************************************************************/
/*   Speicherallokieren fuer lange Zahlen                                   */
/****************************************************************************/

  FBh     = lalloc(5);
  kN      = lalloc(stell);
  N       = lalloc(stell);
  hilf    = lalloc(3 * stel(kN) + 4);
```

```
  X_quad  = lalloc(3 * stel(kN) + 4);
  Y_quad  = lalloc(3 * stel(kN) + 4);

  string_to_l(zeile,kN);

/*****************************************************************************/
/*   Bestimme den verwendeten Multiplier                                     */
/*****************************************************************************/

  for (j = 0; j <= 47; j++)
    {
      lsdiv(kN,cand[j], X_quad, &rest);
      if (rest == 0)
        {
          printf("Multiplier : %d\n",cand[j]);
          llsto(X_quad, N);
        }
    }

  if (!(exponent = (int*)calloc(FB[0] + 2, sizeof(int))))
    return(-1108);

/*****************************************************************************/
/*   Oeffne die Datei mit den full relations, die zum Aufbau des Gleichungs-  */
/*   systems verwendet wurden                                                */
/*****************************************************************************/

  if (!faktorenmatrix)
    return(-2102);

/*****************************************************************************/
/*  Oeffnen der Datei mit den L"osungen des Gleichungssystems                */
/*****************************************************************************/

  if (!(allsolutions = fopen("ALLSOLUTIONS","r")))
    {
      fclose(faktorenmatrix);
      return(1);
    }

/*****************************************************************************/
/*  Bestimme 6 L"osungen die zu echten Teilern f"uhren                       */
/*****************************************************************************/

  while (gefunden < 6)
    {
      set1(X_quad);
      set1(Y_quad);
      zaehler = 1;
      memset(exponent,0,sizeof(int)*(FB[0]+2));
      fseek(faktorenmatrix, 0, 0);

/*****************************************************************************/
/*  Lese L"osung ein                                                         */
/*****************************************************************************/

      while (1)
        {
          if ((eoftest = fscanf(allsolutions,"%d\n", &zeilenindex)) == EOF)
            break;
          if (zeilenindex == 7654321)
            break;

          while (zaehler <= zeilenindex)
            {
              fgets(zeile, 2000, faktorenmatrix);
```

```
            zaehler++;
          }

        sscanf(zeile, "%s", Hstring);
        string_to_l(Hstring, hilf);
        llmul(Y_quad, hilf, Y_quad);
        lldiv(Y_quad, kN, 0, Y_quad);

/******************************************************************************/
/* lese Exponenten und addiere sie auf den bestehende Exponentenvektor auf    */
/******************************************************************************/

        if (pp = strchr(zeile, ':'))
          {
            pp += 2;
            p = strtok(pp, " \n");
            while (p != NULL)
             {
               e = atoi(p);
               if (!e)
                 break;
               p = strtok(NULL, " \n");
               exponent[atoi(p)] += e;
               p = strtok(NULL, " \n");
             }
          }
        if (neu = strchr(zeile, '@'))     /* LARGE PRIME Relation   */
          {
            sscanf(zeile, "%*s %s", Hstring);
            string_to_l(Hstring, hilf);
            llmul(X_quad, hilf, X_quad);
            lldiv(X_quad, kN, 0, X_quad);
          }

      }

    if (eoftest == EOF)
      break;

/******************************************************************************/
/*  Berechne Y                                                                */
/******************************************************************************/

    for (i = 2; i < FB[0] + 2; i++)
      if (exponent[i])
        {
          slsto(FB[i], FBh);
          if (exponent[i] < 0)
            {
              slsto(abs(exponent[i]) >> 1, hilf);
              potenz_mod(FBh, hilf, kN, hilf);
              llmul(Y_quad, hilf, Y_quad);
              lldiv(Y_quad, kN, 0, Y_quad);
            }
          else
            {
              slsto(exponent[i] >> 1, hilf);
              potenz_mod(FBh, hilf, kN, hilf);
              llmul(X_quad, hilf, X_quad);
              lldiv(X_quad, kN, 0, X_quad);
            }
        }

/******************************************************************************/
/*  Berechne (X+Y) und (X-Y)                                                  */
/******************************************************************************/
```

```
        lladd(X_quad, Y_quad, hilf);
        llsub(X_quad, Y_quad, Y_quad);
        llsto(hilf, X_quad);

        lldiv(X_quad, kN, 0, X_quad);
        lldiv(Y_quad, kN, 0, Y_quad);
        llgcd0(X_quad, N, X_quad);
        llgcd0(Y_quad, N, Y_quad);

/******************************************************************************/
/*  Test ob echter Teiler gefunden                                            */
/******************************************************************************/

        if ((llcomp(X_quad, N) == 0) || (llcomp(Y_quad, N) == 0) )
          {
            printf("%d. Loesungsversuch schlug fehl\n", loesung);
            fflush(stdout);
            loesung++;
            fseek(faktorenmatrix, 0, 0);
            memset(exponent, 0, (FB[0] + 2) * sizeof(int));
          }
        else
          {
            lwrite(X_quad);
            lwrite(Y_quad);
            gefunden ++;
            printf(--------------------------------------------------------\n");
          }
      }

  fclose(faktorenmatrix);
  fclose(allsolutions);

  free(X_quad);
  free(Y_quad);
  free(hilf);
  free(FBh);
  free(kN);
  free(N);
  return(0);
}
```

A.2 weitere Module

finde_k.c

```
#include "hmpqs.h"

/*****************************************************************************/
/* DOC                  FINDE_K()                                            */
/*                                                                           */
/*      Die Funktion berechnet einen kleinen Multiplier k                    */
/*                                                                           */
/*                                                                           */
/*      Eingabe:  LONGTYPE *N    : zu faktorisierende Zahl                   */
/*                                                                           */
/*                int bis   : Anzahl der Primzahlen, fuer die die Guete des  */
/*                            Multipliers getestet wird                      */
/*                                                                           */
/*      Ausgabe:  %                                                          */
/*                                                                           */
/*      Return :  int k     : gefundener Multiplier                          */
/*                            oder Fehlercode                                */
/*                                                                           */
/*      zuletzt geaendert am 10.10.92                                        */
/*****************************************************************************/

int finde_k(N, bis)
     LONGTYPE    N;
     int         bis;
{
  LONGTYPE        kN;
  int             prime[600],p;
  FILE            *fp = fopen ("/usr/local/dat/PRIMES", "r");
  register int    j,i=0, k=1, plauf, nmod4, zeiger;
  double          wert, bestwert = 1,plus;
  static long     cand[]= { 1,3,5,7,11,13,15,17,19,21,23,29,31,33,35,37,39,41,
                            43,47,51,53,55,57,59,61,65,67,71,73,79,83,85,89,91,
                            93,95,97,101,103,105,107,109,113,127,131,137,139};

  if (!fp)
    return(-2106);

  kN = lalloc(stel(N) +3);

  while ((fscanf(fp, "%d", &p) != EOF) && (i <= 5*bis))
    prime[i++] = p;
  fclose(fp);
  nmod4 = N[1] & 3;
  for (j = 0; j <= 52; j++)
    {
      if (((p = cand[j]) * nmod4 % 4) != 1)
        continue;
      wert = - 0.7 *  log2((double)p);
      lsmul(N, p, kN);
      if ((kN[1] & 7) == 1)
        wert += 1.38629;

      plauf = prime[0];
      zeiger = 1;
```

```
      i = 0;
      while ((i <= bis) && (zeiger < 5*bis))
        {
          if (legendre(kN,plauf) == 1)
            {
              i++;
              plus = log2((double)plauf)/(double)plauf;
              if ((p % plauf) == 0)
                wert += plus;
              else wert += 2 * plus;
            }
          plauf = prime[zeiger++];
        }
      if (wert > bestwert)
        {
          bestwert = wert;
          k = p;
        }
    }
  free(kN);
  return(k);
}
```

erken_k.c

```
#include "hmpqs.h"

/****************************************************************************/
/* DOC                  Erkenne_k()                                         */
/*                                                                          */
/*      Die Funktion sucht den kleinen Multiplier k in kN                   */
/*                                                                          */
/*                                                                          */
/*      Eingabe:  LONGTYPE kN    : zu faktorisierende Zahl                  */
/*                                                                          */
/*      Return :  int k      : gefundener Multiplier                        */
/*                             oder Fehlercode                              */
/*                                                                          */
/*      zuletzt geaendert am 08.07.92                                       */
/****************************************************************************/

int erken_k(N)
     LONGTYPE N;
{
  LONGTYPE        erg;
  register int    j, i=0;
  int             rest;
  static long     cand[]= { 1,3,5,7,11,13,15,17,19,21,23,29,31,33,35,37,39,41,
                            43,47,51,53,55,57,59,61,65,67,71,73,79,83,85,89,91,
                            93,95,97,101,103,105,107,109,113,127,131,137,139};

  erg = h_get_temp(DEFAULT_LENGTH);

  for (j = 0; j <= 52; j++)
    {
      lsdiv(N,cand[j], erg, &rest);
      if (rest == 0)
        i = j;
```

```
    }
  h_free_temp(erg);
  return(cand[i]);
}
```

timing.c

```
#include <sys/time.h>
#include <sys/resource.h>

static struct rusage buffer;

/******************************************************************************/
/* DOC       start_timing      get_timing         cont_timing                */
/*                                                                            */
/* Die Funktionen dienen der Ermittlung der verbrauchten Rechenzeit.          */
/* Sie wurden von Stefan Duellmann entwickelt.                                */
/*                                                                            */
/* timer[0] enthaelt die User-Zeit, timer[1] die System-Zeit, je in 1/100 sec */
/*                                                                            */
/* Durch start_timing werden die aktuellen Prozess-Zeiten abgespeichert.      */
/* Durch get_timing werden diese von den neuen aktuellen Prozess-Zeiten       */
/* abgezogen, so dass dann genau die zwischen den beiden Aufrufen verbrauchte */
/* Zeit gespeichert ist.                                                      */
/* Durch cont_timing kann eine beendete Zeitmessung fortgefuehrt werden.      */
/*                                                                            */
/* Letzte Aenderung: 16.08.89                                                 */
/******************************************************************************/

int start_timing(timer)
      int timer[2];
{
   getrusage(0,&buffer);
   timer[0] = buffer.ru_utime.tv_sec * 100 + buffer.ru_utime.tv_usec / 10000;
   timer[1] = buffer.ru_stime.tv_sec * 100 + buffer.ru_stime.tv_usec / 10000;
   return(0);
}

int get_timing(timer)
      int timer[2];

{
   getrusage(0,&buffer);
   timer[0] = buffer.ru_utime.tv_sec * 100 + buffer.ru_utime.tv_usec / 10000
              - timer[0] ;
   timer[1] = buffer.ru_stime.tv_sec * 100 + buffer.ru_stime.tv_usec / 10000
              - timer[1] ;
   return(0);
}
int cont_timing(timer)
      int timer[2];

{
   get_timing(timer);
   return(0);
}
```

```
/****************************************************************************/
/* DOC       anf_zeit()         end_zeit()                                  */
/*                                                                          */
/*  Die Funktionen dienen dazu die Laufzeit von Programmteilen festzustellen. */
/*                                                                          */
/****************************************************************************/

static int timer[2];

anf_zeit(flag)
      char flag;
{
    if (flag == 0)
        unlink("ZEIT");
    start_timing(timer);
}

end_zeit(s)
      char *s;
{
    FILE *fp;

    get_timing(timer);
    fp = fopen("ZEIT", "a");
    fprintf(fp, "%10d %10d %s\n", timer[0], timer[1], s);
    fclose(fp);
}
```

Anhang B

rho Funktion

x	1 + x	2 + x	3 + x	4 + x	5 + x
0	1	0.30685285	0.048608408	0.0049109302	0.00035472529
0.01	0.99004967	0.30189014	0.047595539	0.0047908289	0.00034503364
0.02	0.98019737	0.2970012	0.046602368	0.0046735249	0.00033559801
0.03	0.9704412	0.29218482	0.045628521	0.0045589557	0.00032641187
0.04	0.96077929	0.28743982	0.044673634	0.0044470604	0.00031746885
0.05	0.95120984	0.28276504	0.043737351	0.0043377794	0.00030876275
0.06	0.9417311	0.27815935	0.042819322	0.0042310542	0.00030028753
0.07	0.93234136	0.27362165	0.041919203	0.0041268279	0.00029203727
0.08	0.92303896	0.26915086	0.041036659	0.0040250446	0.00028400622
0.09	0.91382231	0.26474593	0.040171361	0.0039256497	0.00027618878
0.1	0.90468983	0.26040581	0.039322987	0.0038285899	0.00026857947
0.11	0.89563999	0.2561295	0.038491219	0.003733813	0.00026117294
0.12	0.88667132	0.251916	0.037675749	0.0036412679	0.000253964
0.13	0.87778238	0.24776436	0.036876273	0.0035509045	0.00024694756
0.14	0.86897175	0.2436736	0.036092492	0.0034626741	0.00024011868
0.15	0.86023807	0.23964282	0.035324115	0.0033765288	0.00023347252
0.16	0.85158001	0.23567109	0.034570855	0.0032924219	0.00022700439
0.17	0.84299626	0.23175752	0.033832433	0.0032103077	0.00022070968
0.18	0.83448557	0.22790125	0.033108572	0.0031301413	0.00021458392
0.19	0.82604671	0.22410141	0.032399002	0.003051879	0.00020862275
0.2	0.81767846	0.22035717	0.03170346	0.0029754779	0.0002028219
0.21	0.80937965	0.2166677	0.031021684	0.0029008962	0.00019717723
0.22	0.80114915	0.2130322	0.030353421	0.0028280929	0.00019168469
0.23	0.79298584	0.20944988	0.02969842	0.0027570279	0.00018634032

x	1 + x	2 + x	3 + x	4 + x	5 + x
0.24	0.78488863	0.20591996	0.029056436	0.0026876619	0.00018114029
0.25	0.77685646	0.20244169	0.028427229	0.0026199565	0.00017608083
0.26	0.76888829	0.19901433	0.027810563	0.0025538743	0.00017115829
0.27	0.76098312	0.19563713	0.027206206	0.0024893784	0.0001663691
0.28	0.75313994	0.19230939	0.026613931	0.0024264328	0.00016170979
0.29	0.7453578	0.18903041	0.026033514	0.0023650026	0.00015717695
0.3	0.73763575	0.18579949	0.025464737	0.0023050531	0.00015276729
0.31	0.72997288	0.18261596	0.024907385	0.0022465508	0.00014847757
0.32	0.72236828	0.17947916	0.024361248	0.0021894627	0.00014430466
0.33	0.71482108	0.17638843	0.023826117	0.0021337566	0.0001402455
0.34	0.7073304	0.17334314	0.02330179	0.0020794009	0.00013629709
0.35	0.69989543	0.17034267	0.022788068	0.0020263648	0.00013245652
0.36	0.69251532	0.16738638	0.022284754	0.0019746182	0.00012872095
0.37	0.68518928	0.16447369	0.021791657	0.0019241314	0.00012508762
0.38	0.67791652	0.161604	0.021308587	0.0018748756	0.00012155384
0.39	0.67069627	0.15877672	0.020835359	0.0018268226	0.00011811696
0.4	0.66352778	0.15599129	0.020371791	0.0017799446	0.00011477443
0.41	0.65641032	0.15324714	0.019917704	0.0017342146	0.00011152374
0.42	0.64934315	0.15054372	0.019472922	0.0016896061	0.00010836248
0.43	0.64232558	0.14788049	0.019037274	0.0016460932	0.00010528825
0.44	0.63535691	0.14525691	0.018610589	0.0016036507	0.00010229875
0.45	0.62843647	0.14267247	0.018192702	0.0015622536	0.000099391732
0.46	0.62156359	0.14012665	0.017783448	0.0015218777	0.000096564988
0.47	0.61473762	0.13761894	0.017382668	0.0014824993	0.000093816382
0.48	0.60795793	0.13514885	0.016990203	0.0014440952	0.000091143827
0.49	0.6012239	0.13271589	0.016605899	0.0014066427	0.00008854529
0.5	0.59453492	0.13031959	0.016229603	0.0013701195	0.000086018791
0.51	0.58789037	0.12795946	0.015861166	0.0013345039	0.0000835624
0.52	0.58128969	0.12563505	0.015500441	0.0012997747	0.000081174235
0.53	0.57473229	0.1233459	0.015147283	0.001265911	0.000078852466
0.54	0.56821761	0.12109156	0.01480155	0.0012328925	0.000076595306
0.55	0.56174509	0.1188716	0.014463103	0.0012006993	0.000074401016
0.56	0.5553142	0.11668558	0.014131806	0.001169312	0.000072267902
0.57	0.54892441	0.11453307	0.013807522	0.0011387114	0.000070194312
0.58	0.54257518	0.11241365	0.01349012	0.001108879	0.00006817864
0.59	0.53626601	0.11032692	0.01317947	0.0010797966	0.000066219318
0.6	0.5299964	0.10827247	0.012875443	0.0010514463	0.000064314821
0.61	0.52376585	0.10624989	0.012577914	0.0010238106	0.000062463662

x	1 + x	2 + x	3 + x	4 + x	5 + x
0.62	0.51757388	0.10425881	0.01228676	0.00099687267	0.000060664394
0.63	0.51142001	0.10229882	0.012001858	0.00097061569	0.000058915607
0.64	0.50530378	0.10036955	0.01172309	0.0009450234	0.000057215927
0.65	0.49922474	0.098470637	0.011450338	0.00092007986	0.000055564018
0.66	0.49318242	0.096601699	0.011183487	0.00089576947	0.000053958577
0.67	0.4871764	0.094762379	0.010922423	0.00087207701	0.000052398336
0.68	0.48120623	0.09295232	0.010667034	0.00084898757	0.00005088206
0.69	0.4752715	0.091171172	0.010417211	0.00082648658	0.000049408546
0.7	0.46937178	0.089418589	0.010172845	0.00080455979	0.000047976624
0.71	0.46350666	0.087694231	0.0099338317	0.00078319327	0.000046585153
0.72	0.45767574	0.085997765	0.0097000655	0.00076237341	0.000045233023
0.73	0.45187862	0.08432886	0.0094714441	0.00074208689	0.000043919155
0.74	0.44611491	0.082687191	0.0092478666	0.00072232072	0.000042642495
0.75	0.44038424	0.081072441	0.0090292338	0.00070306215	0.00004140202
0.76	0.43468622	0.079484293	0.0088154479	0.00068429878	0.000040196732
0.77	0.42902048	0.077922438	0.0086064131	0.00066601844	0.000039025662
0.78	0.42338666	0.076386571	0.008402035	0.00064820926	0.000037887864
0.79	0.41778441	0.074876391	0.0082022206	0.00063085964	0.00003678242
0.8	0.41221336	0.073391603	0.0080068786	0.00061395824	0.000035708434
0.81	0.40667318	0.071931914	0.0078159192	0.00059749398	0.000034665035
0.82	0.40116353	0.070497037	0.007629254	0.00058145604	0.000033651376
0.83	0.39568406	0.069086688	0.0074467959	0.00056583383	0.000032666632
0.84	0.39023446	0.06770059	0.0072684596	0.00055061703	0.00003171
0.85	0.38481439	0.066338467	0.0070941608	0.00053579554	0.000030780699
0.86	0.37942354	0.065000049	0.0069238167	0.0005213595	0.000029877969
0.87	0.3740616	0.063685068	0.0067573459	0.00050729928	0.000029001071
0.88	0.36872825	0.062393261	0.0065946683	0.00049360548	0.000028149284
0.89	0.3634232	0.061124369	0.0064357051	0.00048026891	0.00002732191
0.9	0.35814614	0.059878137	0.0062803785	0.00046728061	0.000026518265
0.91	0.35289679	0.058654312	0.0061286124	0.00045463182	0.000025737689
0.92	0.34767484	0.057452647	0.0059803315	0.00044231399	0.000024979535
0.93	0.34248003	0.056272897	0.0058354621	0.00043031877	0.000024243178
0.94	0.33731206	0.05511482	0.0056939313	0.00041863802	0.000023528006
0.95	0.33217066	0.053978178	0.0055556677	0.00040726379	0.000022833426
0.96	0.32705556	0.052862737	0.0054206008	0.00039618831	0.000022158861
0.97	0.32196649	0.051768267	0.0052886612	0.00038540402	0.00002150375
0.98	0.31690319	0.050694537	0.0051597808	0.00037490353	0.000020867545
0.99	0.31186539	0.049641325	0.0050338925	0.00036467963	0.000020249716

x	6 + x	7 + x	8 + x	9 + x	10 + x
0	0.000019649747	$8.7457015\,10^{-7}$	$3.2320847\,10^{-8}$	$1.0162544\,10^{-9}$	$2.7701925\,10^{-11}$
0.01	0.000019067133	$8.4693688\,10^{-7}$	$3.1245664\,10^{-8}$	$9.809625\,10^{-10}$	$2.6703914\,10^{-11}$
0.02	0.000018501387	$8.2016146\,10^{-7}$	$3.0205761\,10^{-8}$	$9.4688269\,10^{-10}$	$2.5741531\,10^{-11}$
0.03	0.000017952032	$7.9421777\,10^{-7}$	$2.9199997\,10^{-8}$	$9.1397388\,10^{-10}$	$2.4813514\,10^{-11}$
0.04	0.000017418608	$7.6908047\,10^{-7}$	$2.822727\,10^{-8}$	$8.8219629\,10^{-10}$	$2.391865\,10^{-11}$
0.05	0.000016900663	$7.4472499\,10^{-7}$	$2.728651\,10^{-8}$	$8.5151151\,10^{-10}$	$2.3055765\,10^{-11}$
0.06	0.00001639776	$7.2112747\,10^{-7}$	$2.6376681\,10^{-8}$	$8.2188239\,10^{-10}$	$2.2223728\,10^{-11}$
0.07	0.000015909474	$6.9826478\,10^{-7}$	$2.5496782\,10^{-8}$	$7.9327303\,10^{-10}$	$2.1421445\,10^{-11}$
0.08	0.000015435392	$6.7611448\,10^{-7}$	$2.4645841\,10^{-8}$	$7.6564874\,10^{-10}$	$2.0647864\,10^{-11}$
0.09	0.000014975111	$6.546548\,10^{-7}$	$2.3822921\,10^{-8}$	$7.38976\,10^{-10}$	$1.9901966\,10^{-11}$
0.1	0.000014528239	$6.3386461\,10^{-7}$	$2.3027112\,10^{-8}$	$7.1322241\,10^{-10}$	$1.9182772\,10^{-11}$
0.11	0.000014094397	$6.1372343\,10^{-7}$	$2.2257534\,10^{-8}$	$6.8835666\,10^{-10}$	$1.8489333\,10^{-11}$
0.12	0.000013673214	$5.9421137\,10^{-7}$	$2.1513334\,10^{-8}$	$6.6434849\,10^{-10}$	$1.7820737\,10^{-11}$
0.13	0.00001326433	$5.7530917\,10^{-7}$	$2.0793687\,10^{-8}$	$6.4116867\,10^{-10}$	$1.7176102\,10^{-11}$
0.14	0.000012867396	$5.5699811\,10^{-7}$	$2.0097796\,10^{-8}$	$6.1878895\,10^{-10}$	$1.6554577\,10^{-11}$
0.15	0.000012482072	$5.3926006\,10^{-7}$	$1.9424887\,10^{-8}$	$5.9718201\,10^{-10}$	$1.5955341\,10^{-11}$
0.16	0.000012108027	$5.2207742\,10^{-7}$	$1.8774212\,10^{-8}$	$5.763215\,10^{-10}$	$1.5377603\,10^{-11}$
0.17	0.000011744939	$5.0543313\,10^{-7}$	$1.8145047\,10^{-8}$	$5.561819\,10^{-10}$	$1.4820598\,10^{-11}$
0.18	0.000011392495	$4.8931063\,10^{-7}$	$1.7536691\,10^{-8}$	$5.3673859\,10^{-10}$	$1.4283591\,10^{-11}$
0.19	0.000011050392	$4.7369386\,10^{-7}$	$1.6948465\,10^{-8}$	$5.1796777\,10^{-10}$	$1.3765868\,10^{-11}$
0.2	0.000010718334	$4.5856727\,10^{-7}$	$1.6379713\,10^{-8}$	$4.9984644\,10^{-10}$	$1.3266745\,10^{-11}$
0.21	0.000010396033	$4.4391574\,10^{-7}$	$1.5829799\,10^{-8}$	$4.8235238\,10^{-10}$	$1.278556\,10^{-11}$
0.22	0.00001008321	$4.2972462\,10^{-7}$	$1.5298108\,10^{-8}$	$4.6546413\,10^{-10}$	$1.2321673\,10^{-11}$
0.23	$9.7795941\,10^{-6}$	$4.1597972\,10^{-7}$	$1.4784044\,10^{-8}$	$4.4916094\,10^{-10}$	$1.1874468\,10^{-11}$
0.24	$9.48492\,10^{-6}$	$4.0266726\,10^{-7}$	$1.4287031\,10^{-8}$	$4.3342277\,10^{-10}$	$1.1443352\,10^{-11}$
0.25	$9.1989315\,10^{-6}$	$3.8977388\,10^{-7}$	$1.3806513\,10^{-8}$	$4.1823026\,10^{-10}$	$1.1027751\,10^{-11}$
0.26	$8.921379\,10^{-6}$	$3.7728663\,10^{-7}$	$1.3341947\,10^{-8}$	$4.0356469\,10^{-10}$	$1.0627111\,10^{-11}$
0.27	$8.6520198\,10^{-6}$	$3.6519294\,10^{-7}$	$1.2892814\,10^{-8}$	$3.8940801\,10^{-10}$	$1.0240899\,10^{-11}$
0.28	$8.3906182\,10^{-6}$	$3.5348063\,10^{-7}$	$1.2458606\,10^{-8}$	$3.7574274\,10^{-10}$	$9.868601\,10^{-12}$
0.29	$8.1369448\,10^{-6}$	$3.421379\,10^{-7}$	$1.2038834\,10^{-8}$	$3.6255201\,10^{-10}$	$9.5097192\,10^{-12}$
0.3	$7.8907769\,10^{-6}$	$3.3115328\,10^{-7}$	$1.1633027\,10^{-8}$	$3.4981954\,10^{-10}$	$9.1637748\,10^{-12}$
0.31	$7.6518979\,10^{-6}$	$3.2051567\,10^{-7}$	$1.1240724\,10^{-8}$	$3.3752957\,10^{-10}$	$8.8303059\,10^{-12}$
0.32	$7.4200971\,10^{-6}$	$3.102143\,10^{-7}$	$1.0861483\,10^{-8}$	$3.256669\,10^{-10}$	$8.5088667\,10^{-12}$

x	6 + x	7 + x	8 + x	9 + x	10 + x
0.33	$7.19517\,10^{-6}$	$3.0023874\,10^{-7}$	$1.0494875\,10^{-8}$	$3.1421683\,10^{-10}$	$8.1990271\,10^{-12}$
0.34	$6.9769177\,10^{-6}$	$2.9057885\,10^{-7}$	$1.0140485\,10^{-8}$	$3.0316518\,10^{-10}$	$7.9003725\,10^{-12}$
0.35	$6.765147\,10^{-6}$	$2.8122483\,10^{-7}$	$9.7979111\,10^{-9}$	$2.9249822\,10^{-10}$	$7.6125026\,10^{-12}$
0.36	$6.5596699\,10^{-6}$	$2.7216715\,10^{-7}$	$9.4667646\,10^{-9}$	$2.8220272\,10^{-10}$	$7.3350316\,10^{-12}$
0.37	$6.3603039\,10^{-6}$	$2.633966\,10^{-7}$	$9.1466697\,10^{-9}$	$2.7226588\,10^{-10}$	$7.0675873\,10^{-12}$
0.38	$6.1668715\,10^{-6}$	$2.5490422\,10^{-7}$	$8.8372623\,10^{-9}$	$2.6267534\,10^{-10}$	$6.8098106\,10^{-12}$
0.39	$5.9792005\,10^{-6}$	$2.4668135\,10^{-7}$	$8.5381904\,10^{-9}$	$2.5341918\,10^{-10}$	$6.5613552\,10^{-12}$
0.4	$5.7971231\,10^{-6}$	$2.3871958\,10^{-7}$	$8.2491135\,10^{-9}$	$2.4448584\,10^{-10}$	$6.3218871\,10^{-12}$
0.41	$5.6204767\,10^{-6}$	$2.3101076\,10^{-7}$	$7.9697019\,10^{-9}$	$2.3586421\,10^{-10}$	$6.0910842\,10^{-12}$
0.42	$5.4491029\,10^{-6}$	$2.23547\,10^{-7}$	$7.6996369\,10^{-9}$	$2.2754351\,10^{-10}$	$5.8686357\,10^{-12}$
0.43	$5.2828481\,10^{-6}$	$2.1632063\,10^{-7}$	$7.4386099\,10^{-9}$	$2.1951336\,10^{-10}$	$5.6542419\,10^{-12}$
0.44	$5.121563\,10^{-6}$	$2.0932424\,10^{-7}$	$7.1863224\,10^{-9}$	$2.1176373\,10^{-10}$	$5.4476137\,10^{-12}$
0.45	$4.9651023\,10^{-6}$	$2.0255062\,10^{-7}$	$6.9424858\,10^{-9}$	$2.0428491\,10^{-10}$	$5.2484725\,10^{-12}$
0.46	$4.813325\,10^{-6}$	$1.959928\,10^{-7}$	$6.7068207\,10^{-9}$	$1.9706756\,10^{-10}$	$5.0565494\,10^{-12}$
0.47	$4.6660942\,10^{-6}$	$1.8964403\,10^{-7}$	$6.4790568\,10^{-9}$	$1.9010261\,10^{-10}$	$4.8715851\,10^{-12}$
0.48	$4.5232768\,10^{-6}$	$1.8349776\,10^{-7}$	$6.2589329\,10^{-9}$	$1.8338135\,10^{-10}$	$4.6933294\,10^{-12}$
0.49	$4.3847435\,10^{-6}$	$1.7754762\,10^{-7}$	$6.0461958\,10^{-9}$	$1.7689534\,10^{-10}$	$4.5215414\,10^{-12}$
0.5	$4.2503685\,10^{-6}$	$1.7178747\,10^{-7}$	$5.8406011\,10^{-9}$	$1.7063642\,10^{-10}$	$4.3559883\,10^{-12}$
0.51	$4.12003\,10^{-6}$	$1.6621134\,10^{-7}$	$5.6419121\,10^{-9}$	$1.6459675\,10^{-10}$	$4.1964459\,10^{-12}$
0.52	$3.9936093\,10^{-6}$	$1.6081344\,10^{-7}$	$5.4498998\,10^{-9}$	$1.5876871\,10^{-10}$	$4.0426978\,10^{-12}$
0.53	$3.8709913\,10^{-6}$	$1.5558819\,10^{-7}$	$5.2643428\,10^{-9}$	$1.5314496\,10^{-10}$	$3.8945356\,10^{-12}$
0.54	$3.7520641\,10^{-6}$	$1.5053014\,10^{-7}$	$5.085027\,10^{-9}$	$1.4771843\,10^{-10}$	$3.7517581\,10^{-12}$
0.55	$3.636719\,10^{-6}$	$1.4563403\,10^{-7}$	$4.911745\,10^{-9}$	$1.4248228\,10^{-10}$	$3.6141712\,10^{-12}$
0.56	$3.5248506\,10^{-6}$	$1.4089476\,10^{-7}$	$4.7442966\,10^{-9}$	$1.3742988\,10^{-10}$	$3.481588\,10^{-12}$
0.57	$3.4163563\,10^{-6}$	$1.363074\,10^{-7}$	$4.5824878\,10^{-9}$	$1.3255487\,10^{-10}$	$3.353828\,10^{-12}$
0.58	$3.3111364\,10^{-6}$	$1.3186714\,10^{-7}$	$4.4261312\,10^{-9}$	$1.2785108\,10^{-10}$	$3.2307174\,10^{-12}$
0.59	$3.2090944\,10^{-6}$	$1.2756936\,10^{-7}$	$4.2750455\,10^{-9}$	$1.2331256\,10^{-10}$	$3.1120884\,10^{-12}$
0.6	$3.1101364\,10^{-6}$	$1.2340955\,10^{-7}$	$4.1290552\,10^{-9}$	$1.1893356\,10^{-10}$	$2.9977793\,10^{-12}$
0.61	$3.014171\,10^{-6}$	$1.1938336\,10^{-7}$	$3.9879907\,10^{-9}$	$1.1470853\,10^{-10}$	$2.8876342\,10^{-12}$
0.62	$2.9211097\,10^{-6}$	$1.1548655\,10^{-7}$	$3.8516879\,10^{-9}$	$1.1063213\,10^{-10}$	$2.7815027\,10^{-12}$
0.63	$2.8308665\,10^{-6}$	$1.1171505\,10^{-7}$	$3.7199882\,10^{-9}$	$1.0669916\,10^{-10}$	$2.6792398\,10^{-12}$
0.64	$2.7433578\,10^{-6}$	$1.0806489\,10^{-7}$	$3.5927381\,10^{-9}$	$1.0290465\,10^{-10}$	$2.5807057\,10^{-12}$
0.65	$2.6585026\,10^{-6}$	$1.0453222\,10^{-7}$	$3.4697891\,10^{-9}$	$9.9243763\,10^{-11}$	$2.4857657\,10^{-12}$

x	6 + x	7 + x	8 + x	9 + x	10 + x
0.66	$2.576222\,10^{-6}$	$1.0111333\,10^{-7}$	$3.3509978\,10^{-9}$	$9.5711845\,10^{-11}$	$2.3942897\,10^{-12}$
0.67	$2.4964396\,10^{-6}$	$9.7804606\,10^{-8}$	$3.2362253\,10^{-9}$	$9.23044\,10^{-11}$	$2.3061525\,10^{-12}$
0.68	$2.4190811\,10^{-6}$	$9.4602558\,10^{-8}$	$3.1253375\,10^{-9}$	$8.9017086\,10^{-11}$	$2.2212333\,10^{-12}$
0.69	$2.3440743\,10^{-6}$	$9.15038\,10^{-8}$	$3.0182044\,10^{-9}$	$8.5845712\,10^{-11}$	$2.1394156\,10^{-12}$
0.7	$2.2713493\,10^{-6}$	$8.8505055\,10^{-8}$	$2.9147005\,10^{-9}$	$8.2786231\,10^{-11}$	$2.060587\,10^{-12}$
0.71	$2.2008381\,10^{-6}$	$8.5603145\,10^{-8}$	$2.8147045\,10^{-9}$	$7.9834735\,10^{-11}$	$1.9846394\,10^{-12}$
0.72	$2.1324746\,10^{-6}$	$8.2794993\,10^{-8}$	$2.7180989\,10^{-9}$	$7.6987452\,10^{-11}$	$1.9114682\,10^{-12}$
0.73	$2.0661948\,10^{-6}$	$8.0077617\,10^{-8}$	$2.6247703\,10^{-9}$	$7.424074\,10^{-11}$	$1.840973\,10^{-12}$
0.74	$2.0019366\,10^{-6}$	$7.744813\,10^{-8}$	$2.5346089\,10^{-9}$	$7.1591084\,10^{-11}$	$1.7730566\,10^{-12}$
0.75	$1.9396393\,10^{-6}$	$7.4903733\,10^{-8}$	$2.4475086\,10^{-9}$	$6.9035088\,10^{-11}$	$1.7076255\,10^{-12}$
0.76	$1.8792446\,10^{-6}$	$7.2441716\,10^{-8}$	$2.3633666\,10^{-9}$	$6.6569475\,10^{-11}$	$1.6445896\,10^{-12}$
0.77	$1.8206953\,10^{-6}$	$7.0059454\,10^{-8}$	$2.2820838\,10^{-9}$	$6.4191083\,10^{-11}$	$1.5838618\,10^{-12}$
0.78	$1.7639363\,10^{-6}$	$6.7754403\,10^{-8}$	$2.2035643\,10^{-9}$	$6.1896856\,10^{-11}$	$1.5253585\,10^{-12}$
0.79	$1.708914\,10^{-6}$	$6.5524101\,10^{-8}$	$2.1277153\,10^{-9}$	$5.9683846\,10^{-11}$	$1.4689988\,10^{-12}$
0.8	$1.6555763\,10^{-6}$	$6.3366161\,10^{-8}$	$2.054447\,10^{-9}$	$5.7549208\,10^{-11}$	$1.4147048\,10^{-12}$
0.81	$1.6038725\,10^{-6}$	$6.1278272\,10^{-8}$	$1.9836727\,10^{-9}$	$5.5490194\,10^{-11}$	$1.3624015\,10^{-12}$
0.82	$1.5537538\,10^{-6}$	$5.9258196\,10^{-8}$	$1.9153086\,10^{-9}$	$5.3504151\,10^{-11}$	$1.3120165\,10^{-12}$
0.83	$1.5051725\,10^{-6}$	$5.7303763\,10^{-8}$	$1.8492737\,10^{-9}$	$5.1588519\,10^{-11}$	$1.26348\,10^{-12}$
0.84	$1.4580823\,10^{-6}$	$5.5412874\,10^{-8}$	$1.7854895\,10^{-9}$	$4.9740828\,10^{-11}$	$1.2167247\,10^{-12}$
0.85	$1.4124384\,10^{-6}$	$5.3583495\,10^{-8}$	$1.7238803\,10^{-9}$	$4.7958691\,10^{-11}$	$1.1716859\,10^{-12}$
0.86	$1.3681973\,10^{-6}$	$5.1813656\,10^{-8}$	$1.6643727\,10^{-9}$	$4.6239806\,10^{-11}$	$1.1283011\,10^{-12}$
0.87	$1.3253167\,10^{-6}$	$5.0101449\,10^{-8}$	$1.6068961\,10^{-9}$	$4.458195\,10^{-11}$	$1.0865099\,10^{-12}$
0.88	$1.2837556\,10^{-6}$	$4.8445025\,10^{-8}$	$1.5513819\,10^{-9}$	$4.2982978\,10^{-11}$	$1.0462545\,10^{-12}$
0.89	$1.2434742\,10^{-6}$	$4.6842596\,10^{-8}$	$1.4977639\,10^{-9}$	$4.144082\,10^{-11}$	$1.0074787\,10^{-12}$
0.9	$1.204434\,10^{-6}$	$4.5292429\,10^{-8}$	$1.4459781\,10^{-9}$	$3.9953476\,10^{-11}$	$9.701287\,10^{-13}$
0.91	$1.1665973\,10^{-6}$	$4.3792844\,10^{-8}$	$1.3959628\,10^{-9}$	$3.8519017\,10^{-11}$	$9.3415248\,10^{-13}$
0.92	$1.1299279\,10^{-6}$	$4.2342216\,10^{-8}$	$1.347658\,10^{-9}$	$3.7135582\,10^{-11}$	$8.9949992\,10^{-13}$
0.93	$1.0943905\,10^{-6}$	$4.0938972\,10^{-8}$	$1.301006\,10^{-9}$	$3.5801374\,10^{-11}$	$8.6612272\,10^{-13}$
0.94	$1.0599508\,10^{-6}$	$3.9581586\,10^{-8}$	$1.255951\,10^{-9}$	$3.4514657\,10^{-11}$	$8.3397432\,10^{-13}$
0.95	$1.0265755\,10^{-6}$	$3.8268584\,10^{-8}$	$1.2124389\,10^{-9}$	$3.3273759\,10^{-11}$	$8.0300987\,10^{-13}$
0.96	$9.9423242\,10^{-7}$	$3.6998536\,10^{-8}$	$1.1704174\,10^{-9}$	$3.2077063\,10^{-11}$	$7.7318612\,10^{-13}$
0.97	$9.6289027\,10^{-7}$	$3.5770057\,10^{-8}$	$1.1298362\,10^{-9}$	$3.092301\,10^{-11}$	$7.4446139\,10^{-13}$
0.98	$9.3251868\,10^{-7}$	$3.4581807\,10^{-8}$	$1.0906465\,10^{-9}$	$2.9810096\,10^{-11}$	$7.167955\,10^{-13}$
0.99	$9.0308816\,10^{-7}$	$3.343249\,10^{-8}$	$1.0528011\,10^{-9}$	$2.8736868\,10^{-11}$	$6.9014976\,10^{-13}$

Literaturverzeichnis

[Ber] Franz-Dieter Berger : *Faktorisieren mit elliptischen Kurven*, Diplomarbeit , Universität des Saarlandes 1993

[BreRie] R.P. Brent, H.J.J. te Riele : *Factorizations of* $a^n \pm 1$, $13 \leq a \leq 100$, CWI Report, Department of Numerical Mathematics NM-R9212 June, 1992

[Brent] R.P. Brent : *Some Integer Factorization Algorithms using Elliptic Curves*, Australian Computer Science Communications 8 (1986), S. 149-163

[Bre] David M. Bressoud : *Factorizations and Primalty Testing*, Springer Verlag, 1989

[BucMül] Johannes Buchmann, Volker Müller : *Algorithms for factoring integers*, Skript zur Vorlesung im WS 88/89, 1991

[Coh] H. Cohen : *A Course in Computational Algebraic Number Theory*, Springer Verlag , 1991

[Dül] Stephan Düllmann : *Ein Programmpaket zur langen ganzzahligen Arithmetik*, Dokumentation, 1990

[Knu] Donald E. Knuth : *The Art of Computer Programming*, Vol. 2 Seminumerical Algorithms, Addison Wesley

[KnuPar] Donald E. Knuth, Luis Trabb Pardo : *Analysis of a simple Factorization Algorithm*, Theoretical Computer Science 3 (1976), S. 321-348

[Kob] Neal Koblitz : *A Course in Number Theory and Cryptography*, Springer Verlag, 1987

[Leh] Derrick N. Lehmer : *On the history of the problem of seperating a number into its prime factors.* The scientific monthly, September 1918

[LenMan] Arjen K. Lenstra, M.S. Manasse : *Factoring with two large primes*, preprint, 1992

[Leu] J. van Leuven (Hrsg.) : *Handbook of theoretical computer science*, Volume A, Algorithms and complexity, Elsevier, Amsterdam, 1990

[LIPS] Ralf Roth, Thomas Setz : *LIPS - A System For Distributed Processing On Workstations*, Dokumentation (11.11.1992)

[MehTsa] K. Mehlhorn, A.Tsakalidis : *Data structures*, Kapitel 6 in [Leu]

[NasShn] I. Nassi, B. Shneidermann : *Flowchart techniques for structured programming*, SIGPLAN Notice 8 (1973), No.8, S. 12-26

[Pom] Carl Pommerance : *Factoring*, Cryptology and Computational Number Theory, Proceedings of Symposia in Applied Mathematics, Vol. 42, S. 27 - 47 1990

[Rie] Hans Riesel : *Prime Numbers and Computer Methods for Factorization*, Birkhäuser Verlag, 1985

[RosSch] J. Barkley Rosser, Lowell Schoenfeld : *Approximate Formulas for some Functions of Prime Numbers* , Illinois Journal of Mathematics, vol. 6, S. 64-94, 1985

[Sil] Robert D. Silvermann : *The Multiple Polynomial Quadratic Sieve*, Mathematics of Computation, Vol 48, S. 329 - 339, 1987