Modifikationen zum Number Field Sieve

Diplomarbeit

von

Michael Hürter

in Informatik

Juli 1994

Die Arbeit wurde nach einem Thema von *Prof. Dr. Johannes Buchmann* am Fachbereich 14 der Informatik an der Universität des Saarlandes in Saarbrücken verfaßt.

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

Saarbrücken, 6. Juli 1994

Inhaltsverzeichnis

Kapitel 1

Herleitung

1.1 Einleitung

Das Number Field Sieve, ein Faktorisierungsalgorithmus nach Lenstra, Lenstra, Manasse und Pollard ([9]) und Buhler, Lenstra und Pomerance ([3]) benötigt zur Faktorisierung einer natürlichen Zahl N als Laufzeit

$$e^{(1.923+o(1))(\log N)^{1/3}(\log\log N)^{2/3}}.$$

Es ist damit der asymptotisch schnellste Algorithmus zum Faktorisieren natürlicher Zahlen. Coppersmith zeigt in [5] eine Möglichkeit, diese Laufzeit weiter zu verbessern. Diese Variante soll in Theorie und Praxis durchleuchtet werden. Außerdem soll das von Pollard in [13] vorgestellte Lattice Sieve beschrieben und seine Laufzeit analysiert werden.

Zu Beginn werden einige grundlegende Definitionen zusammengestellt.

1.1. Definition *Bezeichne mit $\mathbb{P}$ die Menge der Primzahlen.*

1.2. Definition *Für eine Menge M definiert man $\#M$ als die Anzahl der Elemente von M.*

1.3. Definition *Für $p \in \mathbb{P}$ und $x \in \mathbb{Z}$ definiert man $\mathrm{ord}_p(x)$ als das größte $e \in \mathbb{N} \cup \{0\}$, für das p^e Teiler von x ist.*

1.2 NFS mit mehreren Polynomen

Zuerst soll die Idee der Faktorisierung einer Zahl N mit dem Number Field Sieve unter Benutzung eines einzigen Polynoms beschrieben werden. Man wählt sich dazu ein normiertes $f(x) \in \mathbb{Z}[x]$ und ein $m \in \mathbb{Z}$, so daß gilt:

$$f(m) \equiv 0 \bmod N.$$

Sei $\rho \in \mathbb{C}$ eine komplexe Nullstelle von $f(x)$, dann existiert folgender Ringhomomorphismus:

$$\varphi : \mathbb{Z}[\rho] \to \mathbb{Z}/N\mathbb{Z} \quad \text{mit} \quad \varphi(\rho) = m \bmod N.$$

Weiterhin sei P eine Menge von Primzahlen. Ziel des Algorithmus ist es, eine Menge S von Paaren $(a,b) \in \mathbb{Z}^2$ zu finden, so daß gilt:

- $\prod\limits_{(a,b)\in S} (a+bm) = \prod\limits_{p\in P} p^{2e_p} = x^2 \quad x = \prod\limits_{p\in P} p^{e_p}$
- $\prod\limits_{(a,b)\in S} (a+b\rho) = \beta^2$, wobei $\beta \in \mathbb{Z}[\rho]$

Aus diesen Paaren (a,b) ergibt sich dann die Kongruenz

$$x^2 \equiv \varphi(\beta)^2 \bmod N,$$

die zu den Teilern der Zahl N führt. Siehe genaueres dazu in Kapitel 5.

Bei der Benutzung mehrerer Polynome geht man ähnlich vor. Hier betrachtet man mehrere Polynome $f_i(x) \in \mathbb{Z}[x]$ und $m_i \in \mathbb{Z}$, so daß gilt:

$$f_i(m_i) \equiv 0 \bmod N.$$

Sei $\rho_i \in \mathbb{C}$ eine komplexe Nullstelle des Polynoms f_i, dann kann man wieder folgende Ringhomomorphismen betrachten:

$$\varphi_i : \mathbb{Z}[\rho_i] \to \mathbb{Z}/N\mathbb{Z} \quad \text{mit} \quad \varphi_i(\rho_i) = m_i \bmod N.$$

Ziel des Algorithmus ist es, für jedes Polynom eine Menge S_i von Paaren $(a,b) \in \mathbb{Z}^2$ zu finden, so daß gilt:

- $\prod\limits_i \prod\limits_{(a,b)\in S_i} (a+bm_i) = \prod\limits_{p\in P} p^{2e_p} = x^2, \quad x = \prod\limits_{p\in P} p^{e_p}$
- $\prod\limits_{(a,b)\in S_i} (a+b\rho_i) = \beta_i^2$, wobei $\beta_i \in \mathbb{Z}[\rho_i], \forall i$

Daraus ergibt sich dann die Kongruenz

$$x^2 \equiv \left(\prod_i \varphi_i(\beta_i)\right)^2 \bmod N.$$

In der Praxis muß man das Produkt $\prod_{(a,b)\in S}(a+b\rho)$ mit $f'(\rho)^2$ multiplizieren, da erst dadurch folgendes sichergestellt ist: Wenn

$$\prod_{(a,b)\in S}(a+b\rho)\cdot f'(\rho)^2 = \beta^2 \text{ mit } \beta \in \mathbb{Q}[\rho],$$

dann ist $\beta \in \mathbb{Z}[\rho]$. Dieses ist in [3] Seite 15 exakt begründet.

Wann ist

$$f'(\rho)^2 \cdot \prod_{(a,b)\in S}(a+b\rho) = \beta^2 \quad ?$$

Um diese Frage zu entscheiden, definiert man für alle (p, cp) mit $f(cp) \equiv 0 \bmod p$

$$e_{p,cp}(a,b) = \begin{cases} \operatorname{ord}_p \mathrm{N}(a+b\rho) & : \text{ falls } a+b\cdot cp \equiv 0 \bmod p \\ 0 & : \text{ sonst} \end{cases}$$

$$q_{p,cp}(a,b) = \begin{cases} 0 & : \text{ falls } a+b\cdot cp \text{ quadratischer Rest modulo } p \\ 1 & : \text{ sonst} \end{cases}$$

Damit man ein Quadrat erhält, muß folgendes gelten:

$$\sum_{(a,b)\in S} e_{p,cp}(a,b) \equiv 0 \bmod 2 \qquad \forall_{(p,cp)}$$

Dies reicht jedoch noch nicht aus. Man kann erst sicher sein, wenn man etwa $\log N$ Paare (q, cq), die

- $f(cq) \equiv 0 \bmod q$
- $a + b\cdot cq \not\equiv 0 \bmod q$

erfüllen, betrachtet, so daß folgendes gilt:

$$\sum_{(a,b)\in S} q_{q,cq}(a,b) \equiv 0 \bmod 2 \qquad \forall_{(q,cq)}$$

Näheres dazu steht in [1].

Wie erhält man die Paare (a, b) ?
Man wählt einen Bereich, in dem die (a, b) liegen sollen und knüpft an die Paare (a, b) weitere Bedingungen.
Definiere dazu Faktorbasen

- die rationale Faktorbasis
  $FBR = \{\text{ersten } B_1' \text{ Primzahlen}\} = \{p \in \mathbb{P} \mid p \leq B_1\}$
- die algebraischen Faktorbasen
  $FBA_i = \{\text{ersten } B_2' \text{ Paare } (p, cp) \mid f_i(cp) \equiv 0(p)\} = \{(p_{i0}, cp_{i0}), \ldots, (p_{iB_2'-1}, cp_{iB_2'-1})\}$
- die quadratischen Faktorbasen
  $FBQ_i = \{B_3' \text{ Paare } (p, cp) \mid f_i(cp) \equiv 0(p)\} = \{(q_{i0}, cq_{i0}), \ldots, (q_{iB_3'-1}, cq_{iB_3'-1})\}$

Sei B_2 das größte p in FBA_i.

Für ein spezielles Polynom sei FBA die algebraische Faktorbasis. Dann geht man so vor:
Suche Paare (a, b) mit

- $\gcd(a, b) > 1$
- $a + bm = lp \cdot \prod_{i=0}^{B_1'-1} p_i^{e_i} \quad \text{mit } lp \in \mathbb{P}$
- $e_{p,cp}(a, b) > 0$ für höchstens zwei $(p, cp) \notin FBA$.

Kombiniere aus diesen (a, b) Mengen M für das Polynom f. Dabei gilt für $M = \{(a, b)\}$

- $\prod_{(a,b)\in M} (a + bm) = lp(M)^2 \cdot \prod_{i=0}^{B_1'-1} p_i^{e_i} \quad \text{mit } lp(M) \in \mathbb{N}$
- $\sum_{(a,b)\in M} e_{p,cp}(a, b) \equiv 0 \bmod 2 \quad \text{für alle} \quad (p, cp) \notin FBA$

Dies macht man für jedes Polynom und erhält so Mengen M_{ij}. Um zu einem linearen Gleichungssystem über F_2 zu kommen, führt man die Variablen x_{ij} ein, die folgende Bedeutung haben:

$$x_{ij} = \begin{cases} 1 & : \text{ falls Menge } M_{ij} \text{ benutzt} \\ 0 & : \text{ sonst} \end{cases}$$

Sie bilden die Variablen des Systems. Folgende Vektoren bilden die zum linearen Gleichungssystem gehörende Matrix:

$$\begin{aligned} r_{ijk} &= \sum_{(a,b)\in M_{ij}} \operatorname{ord}_{p_k}(a + bm_i) \\ a_{ijk} &= \sum_{(a,b)\in M_{ij}} e_{p_{ik},cp_{ik}}(a, b) \\ q_{ijk} &= \sum_{(a,b)\in M_{ij}} q_{q_{ik},cq_{ik}}(a, b) \end{aligned}$$

Hierbei geht k bei r_{ijk} von 0 bis $B_1'-1$, bei a_{ijk} von 0 bis $B_2'-1$ und bei q_{ijk} von 0 bis $B_3'-1$.

Es muß gelten:

$$\begin{array}{rrcl} & \prod_{i,j}\left(\prod_{(a,b)\in M_{ij}}(a+bm_i)\right)^{x_{ij}} & = & x^2 \\ \forall_i & f'(\rho_i)^2\cdot\prod_j\left(\prod_{(a,b)\in M_{ij}}(a+b\rho_i)\right)^{x_{ij}} & = & \beta_i^2 \end{array}$$

Unter der Verwendung der definierten Vektoren führen diese Bedingungen zu einem linearen Gleichungssystem, das durch folgende Gleichungen definiert ist:

$$\begin{array}{rrcl} & \sum_{i,j} r_{ij}\cdot x_{ij} & \equiv & 0 \bmod 2 \\ \forall_i & \sum_j a_{ij}\cdot x_{ij} & \equiv & 0 \bmod 2 \\ \forall_i & \sum_j q_{ij}\cdot x_{ij} & \equiv & 0 \bmod 2 \end{array}$$

Sei

$$y = (x_{01},\ldots,x_{11},\ldots,x_{n-1,1},\ldots).$$

Dann ist dies ist ein Gleichungssystem der Form $A\cdot y = 0$, wobei die Matrix $B_1' + n\cdot B_2' + n\cdot B_3'$ Zeilen hat und n die Anzahl der verwendeten Polynome ist. Damit es mindestens eine Lösung gibt, muß man für jedes Polynom $\frac{B_1'}{n}+B_2'+B_3'$ Mengen M_{ij} finden. Man erhält dann für eine Lösung x_{ij}

$$\begin{array}{rcl} x & = & \prod_i f_i'(m_i)\cdot\prod_{x_{ij}=1} lp(M_{ij})\cdot\prod_{k=0}^{B_1'-1} p_k^{\left(\sum_{x_{ij}=1} r_{ijk}\right)/2} \\ \beta_i^2 & = & f_i'(\rho_i)^2\cdot\prod_{x_{ij}=1}\prod_{(a,b)\in M_{ij}}(a+b\rho_i) \end{array}$$

1.3 Die Laufzeit

In diesem Abschnitt geht es um die theoretische Laufzeit, insbesondere um die Verbesserung bei der Benutzung mehrerer Polynome. Coppersmith hat in [5] gezeigt, daß bei optimaler Wahl der Parameter die asymptotische Laufzeit von $L\left[\frac{1}{3}, 1.923\right]$ auf $L\left[\frac{1}{3}, 1.902\right]$ verbessert wird. Dabei werden allerdings Varianten wie Benutzen von Largeprimes nicht berücksichtigt. Im folgenden wird gezeigt, wie die Parameter zu wählen sind, und warum die Laufzeit dadurch besser wird.

1.3.1 Grundlagen

Es erweist sich als unumgänglich, Zahlen als Werte folgender Funktion darzustellen:

$$L[\alpha, \beta] = L_N[\alpha, \beta] = e^{(\beta+o(1))(\log N)^\alpha (\log\log N)^{1-\alpha}}$$

Für diese Funktion gelten interessante Eigenschaften, besonders im Zusammenhang mit der Zerlegung natürlicher Zahlen in Primfaktoren.

1.4. Definition *Eine natürliche Zahl z heißt y-smooth, wenn kein Primteiler von z größer als y ist.*
Mit $\Psi(x, y)$ bezeichnet man die Anzahl der Zahlen z mit $1 \leq z \leq x$, die y-smooth sind.

Stellt man Zahlen als Werte der Funktion L dar, kann man nähere Aussagen machen.

1.5. Satz *Sei $x = L[a, b]$ und $y = L[c, d]$ mit $a > c$ und $b, d > 0$, dann gilt:*

$$\frac{1}{x}\Psi(x, y) = 1/L[a-c, b(a-c)/d].$$

1.3.2 Laufzeit bei einem Polynom

Als Einstieg soll nun die Laufzeit bei der Benutzung eines Polynomes betrachtet werden. Man wählt eine Schranke $E \in \mathbb{N}$, so daß nur Paare (a, b) mit

$$|a| \leq E \text{ und } 1 \leq b \leq E$$

betrachtet werden. Es wird gezeigt, daß bei der Benutzung folgender Parameter der Algorithmus das gewünschte Ergebnis liefert:

- $d = \delta \left(\frac{\log N}{\log\log N}\right)^{\frac{1}{3}}, \delta = 3^{\frac{1}{3}}$
- $m = N^{1/d} = L\left[\frac{2}{3}, \frac{1}{\delta}\right]$
- für die Koeffizienten des Polynoms gilt: $|c_i| \leq L\left[\frac{2}{3}, \frac{1}{\delta}\right]$
- $B_1 = L\left[\frac{1}{3}, \beta\right], \beta = \frac{2}{3^{\frac{2}{3}}} = 0.961$
- $B_2 = L\left[\frac{1}{3}, \gamma\right], \gamma = \frac{2}{3^{\frac{2}{3}}} = 0.961$

- $E = L\left[\frac{1}{3}, \epsilon\right], \epsilon = \frac{2}{3^{\frac{2}{3}}} = 0.961$

Um zu zeigen, daß der Algorithmus funktioniert, muß man genügend Paare (a, b) mit $|a| \leq E, 1 \leq b \leq E$ finden, die folgende Eigenschaften erfüllen:

- $\gcd(a, b) = 1$
- $a + bm$ ist B_1-smooth
- $\mathrm{N}(a + b\rho)$ ist B_2-smooth

Für die Norm von $a + b\rho$ gilt:

$$\mathrm{N}(a + b\rho) = a^d - c_{d-1}a^{d-1}b + \ldots + (-1)^d c_0 b^d.$$

Hierbei erhält man für die Größenordnung von $a+bm$ und $\mathrm{N}(a+b\rho)$ folgende Werte:

- $a + bm = mE = L\left[\frac{2}{3}, \frac{1}{\delta}\right]$
- $\mathrm{N}(a + b\rho) = dmE^d = L\left[\frac{2}{3}, \frac{1}{\delta}\right] L\left[\frac{2}{3}, \epsilon\delta\right] = L\left[\frac{2}{3}, \epsilon\delta + \frac{1}{\delta}\right]$

Dadurch ist es einfach abzuschätzen, wieviele Paare (a, b) die Bedingungen erfüllen. Ihre Anzahl ist nämlich

$$E^2 \frac{1}{mE}\Psi(mE, B_1)\frac{1}{dmE^d}\Psi(dmE^d, B_2) = \frac{L\left[\frac{1}{3}, 2\epsilon\right]}{L\left[\frac{1}{3}, \frac{1}{3\delta\beta}\right] L\left[\frac{1}{3}, \frac{\epsilon\delta+1/\delta}{3\gamma}\right]} = L\left[\frac{1}{3}, 2\epsilon - \frac{1}{3\delta\beta} - \frac{\epsilon\delta}{3\gamma} - \frac{1}{3\delta\gamma}\right].$$

Dieser Wert ist bei der Wahl der Parameter ungefähr B_1 und somit auf jeden Fall größer als $B_1' + B_2' + \log(N)$.

Die Laufzeit des gesamten Algorithmus hängt von den Laufzeiten folgender Teile ab:

- Sieben über einen Bereich, der durch E definiert ist
- Lösen eines linearen Gleichungssystems über F_2, dessen Größe durch B_1 und B_2 bestimmt ist
- Ziehen der Wurzel einer algebraischen Zahl, wobei die Komplexität dieses Problems von B_2 abhängt

Bei dieser Parameterwahl ergeben sich folgende Laufzeiten:

- für den Siebalgorithmus $E^2 = L\left[\frac{1}{3}, 2\epsilon\right]$
  Diese Zeit ergibt sich aus der Anzahl der zu untersuchenden Paare (a, b), die E^2 beträgt.

- für das Lösen des Systems $B_1^2 = L\left[\frac{1}{3}, 2\epsilon\right]$
  Die zu dem linearen Gleichungssystem gehörende Matrix hat ungefähr B_1 Zeilen und Spalten. Daraus ergibt sich als Zeit zum Lösen des Systems B_1^2.

- für das Ziehen der Wurzel $B_2^2 = L\left[\frac{1}{3}, 2\epsilon\right]$
  Die Zeit, die benötigt wird, um die Wurzel aus einer algebraischen Zahl zu ziehen, wird in [5] als B_2^2 angegeben.

Als Gesamtlaufzeit ergibt sich also $L\left[\frac{1}{3}, 2\epsilon\right]$.

1.3.3 Laufzeit bei mehreren Polynomen

Bei der Wahl mehrerer Polynome läßt sich diese Zeit asymptotisch verbessern. Dabei wählt man für die Anzahl

$$\frac{B_1'}{B_4'} = L\left[\frac{1}{3}, \beta - \gamma\right],$$

wobei $B_1' = \pi(B_1)$ und $B_4' = \pi(B_2) + O(logN)$.

Für jedes Polynom müssen $2 \cdot B_4'$ Paare (a, b) gefunden werden, so daß sich analog zu dem Fall eines Polynoms folgende Bedingung an die Parameter ergibt:

$$2\epsilon - \frac{1}{3\delta\beta} - \frac{\epsilon\delta}{3\gamma} - \frac{1}{3\delta\gamma} \geq \gamma.$$

Man findet heraus, daß sich die optimale Laufzeit bei der Wahl folgender Parameter ergibt:

$$\epsilon = \beta = \left(\frac{46 + 13\sqrt{13}}{108}\right) = 0.95094$$

$$\gamma = \epsilon \times \left(\frac{\sqrt{13} - 1}{3}\right) = 0.82591$$

$$\delta = \epsilon \times \left(\frac{4\sqrt{13} - 10}{3}\right) = 1.40175$$

In diesem Fall ergibt sich für die Gesamtlaufzeit $L\left[\frac{1}{3}, 1.902\right]$. Diese Zeit resultiert aus folgenden Teilen:

- Bestimmen der Paare (a, b).
  Hierfür gibt Coppersmith in [5] auf Seite 9 den Wert $L\left[\frac{1}{3}, 2\epsilon\right]$ an.
- Lösen eines linearen Gleichungssystems, wobei die zugehörige Matrix $2 \cdot B_1'$ Zeilen hat. Als Zeit ergibt sich hierfür $L\left[\frac{1}{3}, 2\beta\right]$.
- Ziehen der Wurzel aus einer algebraischen Zahl für jedes verwendete Polynom braucht Zeit $\frac{B_1'}{B_4'} \cdot B_2^2 = L\left[\frac{1}{3}, \beta + \gamma\right]$.

1.4 Nichtnormierte Polynome

Durch die Verwendung nichtnormierter Polynome läßt sich die Laufzeit des Algorithmus wesentlich verbessern. Das hängt damit zusammen, daß man den Parameter m viel kleiner wählen kann, was vor allem Auswirkungen auf die Koeffizienten der Polynome (in der Größenordnung von m) und damit auf die Größe der Norm hat. Diese Verbesserung läßt sich ohne größeren Aufwand realisieren und soll nun näher erläutert werden.

1.4.1 Wahl der Polynome

Man kann die Polynome so wählen:

- setze $m = \lceil \sqrt[d+1]{N} \rceil$
- berechne die $m-$adische Darstellung von N und daraus das Polynom $f(x) = c_d x^d + \ldots + c_0$

1.4.2 Unterschiede

Sei ein Polynom $f(x) = c_d x^d + \ldots + c_0$ gegeben, $\rho \in \mathbb{C}$ eine komplexe Nullstelle. Dann definiert man dazu ein normiertes Polynom $g(x)$ durch

$$g(x) = f\left(\frac{x}{c_d}\right) \cdot c_d^{d-1}.$$

Definiert man weiter

$$\mathrm{N}'(a, b) := f\left(-\frac{a}{b}\right) \cdot (-b)^d = c_d a^d - c_{d-1} a^{d-1} b + \ldots + (-1)^d c_0 b^d,$$

und setzt $\omega = c_d \cdot \rho$, so gilt folgendes:

- $g(\omega) = g(c_d \cdot \rho) = f\left(\frac{c_d \cdot \rho}{c_d}\right) \cdot c_d^{d-1} = f(\rho) \cdot c_d^{d-1} = 0$

- $\mathbb{Q}[\rho] = \mathbb{Q}[\omega]$
- $\mathbb{Z}[\omega]$ ist ein kommutativer unitärer Ring mit folgendem Ringhomomorphismus $\varphi : \mathbb{Z}[\omega] \to \mathbb{Z}/N\mathbb{Z}$ mit
$\varphi(\omega) = c_d \cdot m \bmod N$.
- $\mathrm{N}(c_d a + b\omega) = \mathrm{N}'(a, b) \cdot c_d^{d-1}$

Damit sieht die Herleitung für die Benutzung mehrerer nichtnormierter Polynome so aus: Seien $f_i(x)$ die benutzten Polynome mit Nullstellen ρ_i, $g_i(x)$ die zugehörigen normierten Polynome mit Nullstellen ω_i.
$f_i(x)$ habe die Darstellung $f_i(x) = c_{id}x^d + \ldots + c_{i0}$.
Zu jedem Polynom existiert natürlich auch ein Homomorphismus φ_i.

Man sucht nun eine Menge S_i von Paaren (a, b) für jedes Polynom f_i. Genauso wie im Fall normierter Polynome muß gelten:

$$\prod_i \prod_{(a,b)\in S_i} (a + bm_i) = x^2.$$

Auf der algebraischen Seite muß für jedes Polynom f_i gelten:

$$g_i'(\omega_i)^2 \cdot \prod_{(a,b)\in S_i} (ac_{id} + b\omega_i) = \beta_i^2, \text{wobei } \beta_i \in \mathbb{Z}[\omega_i]$$

Dann gilt modulo N:

$$\begin{aligned}
\left(\prod_i \varphi_i(\beta_i)\right)^2 &= \prod_i \varphi_i\left(g_i'(\omega_i)^2 \prod_{(a,b)\in S_i} (c_{id}a + b\omega_i)\right) \\
&\equiv \prod_i \left(g_i'(c_{id}m_i)^2 \prod_{(a,b)\in S_i} (c_{id}a + b \cdot c_{id}m_i)\right) \\
&\equiv \prod_i \left(c_{id}^{d-2} f_i'(m_i)^2 c_{id}^{\#S_i} \prod_{(a,b)\in S_i} (a + bm_i)\right) \\
&\equiv \left(\prod_i f_i'(m_i) c_{id}^{\#S_i/2+d-2} \cdot x\right)^2
\end{aligned}$$

Ausgehend vom Fall normierter Polynome kann man mit wenigen Veränderungen in den einzelnen Algorithmen auch nichtnormierte Polynome benutzen. Man muß lediglich folgendes ändern:

- Beachte die Primzahlen p, die c_d teilen. Man nimmt für sie das Paar (p, p) zusätzlich in die algebraische Faktorbasis auf. In der Siebphase und in der Zerlegung gilt dann, daß p den Wert $N'(a, b)$ teilt, falls p Teiler von b ist.

- Für jedes Polynom muß die Anzahl der dazu gehörenden Paare (a, b) gerade sein. Das bedeutet, daß die Anzahl der Mengen M_{ij} mit ungerader Elementanzahl gerade sein muß. Um das zu erzwingen, führt man folgenden Wert für jede M_{ij} ein:

  $$g_{ij} = \begin{cases} 1 & : \text{ falls Elementanzahl von } M_{ij} \text{ ungerade} \\ 0 & : \text{ sonst} \end{cases}$$

  Dann muß für jedes Polynom f_i zusätzlich folgende Gleichung erfüllt sein:

  $$\sum_j x_{ij} \cdot g_{ij} \equiv 0 \bmod 2$$

  Man hat also pro Polynom eine Zeile mehr in der Matrix, was bedeutet, daß man auch eine weitere Menge M_{ij} braucht.

- Für die Norm von $ac_d + b\omega$ gilt:

  $$\mathrm{N}(ac_d + b\omega) = c_d^{d-1} \cdot \mathrm{N}'(a, b).$$

- Bei der Berechnung der Teiler der Zahl N ist weiter zu beachten, daß die Werte $c_d^{\#S}$ und c_d^{d-2} aufmultipliziert werden, was in der Herleitung oben genau beschrieben ist.

Kapitel 2

Das Lattice Sieve

2.1 Der Algorithmus

Das Lattice Sieve ist das Herzstück des gesamten Algorithmus. Es bestimmt für jedes Polynom Paare $(a, b) \in \mathbb{Z}^2$, die dann untersucht werden, ob man aus ihnen die vorher beschriebenen Mengen M_{ij} gewinnen kann. Diese Paare (a, b) erfüllen gewissse Eigenschaften, aber ob sie wirklich nützlich sind, muß durch Zerlegen später festgestellt werden. Zuerst werden die Parameter und die Faktorbasen berechnet. Man erhält so:

- $FBR = \{\text{ersten } B_1' \text{ Primzahlen}\} = \{p \in \mathbb{P} \mid p \leq B_1\}$
- $FBA_i = \{\text{ersten } B_2' \text{ Paare } (p, cp) \mid f_i(cp) \equiv 0(p)\} = \{(p_{i0}, cp_{i0}), \ldots, (p_{iB_2'-1}, cp_{iB_2'-1})\}$

Von allen Paaren (a, b) sind solche herauszusieben, für die gilt:

- $\gcd(a, b) = 1$
- $a + bm$ ist B_1-smooth
- $\mathrm{N}(a + b\rho)$ ist B_2-smooth

Solche Paare stellen eine Menge M_{ij} dar. Als Variante ist es auch nützlich, weitere Paare (a, b) zu bestimmen, die die Eigenschaft erfüllen, daß $a + bm$ einen Primfaktor (Largeprime) hat, der kleiner als eine Schranke MAXLARGE ist. Außerdem darf entweder

- (Double Large Prime Variante) $\mathrm{N}(a + b\rho)$ auch einen Primfaktor bis zu einer Schranke MAXPILARGE oder

- (Triple Large Prime Variante) $\mathrm{N}(a+b\rho)$ zwei Primfaktoren bis zu einer Schranke MAXPILARGE haben.

Der Siebalgorithmus kann zwar solche Paare (a, b) nicht bestimmen, aber es ist möglich, diejenigen Paare (a, b) herauszusieben, die eine gute Chance haben. Es ist offensichtlich, daß je kleiner die Werte a, b sind, desto größer die Chance ist, ein günstiges Paar (a, b) zu erhalten. Genaueres dazu steht in Kapitel 2.6.

Es ist klar, daß man für alle zu betrachtenden (a, b)

- alle Primteiler von $a + bm$, die in der rationalen Faktorbasis sind, bestimmen muß
- alle Primteiler p von $\mathrm{N}(a + b\rho)$ mit $p < B_2$ bestimmen muß

Dabei geht man so vor: Teile die rationale Faktorbasis FBR in zwei Teile auf.

- Small Primes = S
- Medium Primes = M

Dabei ist S die Menge der Elemente von FBR bis zu einer gewissen Grenze und M die Menge der Elemente ab dieser Grenze. Fixiere $q \in M$ fest und verfahre weiter wie folgt:
Es werden alle Paare (a, b) gesucht, die die oben genannten Eigenschaften erfüllen und für die q der größte Primfaktor von $a + bm$ ist.

Bemerkung: Macht man dies für alle $q \in M$, so werden alle Paare (a, b), für die $a + bm$ keinen Primfaktor in M hat, nicht betrachtet. Dies führt zu einer erheblichen Laufzeitverbesserung des Lattice Sieve gegenüber dem konventionellen Siebalgorithmus, was später genau erkärt wird.

Zunächst betrachtet man alle Paare (a, b) mit

$$a + bm \equiv 0 \bmod q.$$

Sie bilden ein zweidimensionales Gitter $L \in \mathbb{Z}^2$. Sei $\{(a_1, b_1), (a_2, b_2)\}$ eine Gitterbasis, so besitzt jedes $(a, b) \in L$ eine Darstellung

$$(a, b) = c \cdot (a_1, b_1) + d \cdot (a_2, b_2).$$

Jedem Paar (a, b) entspricht also ein Paar (c, d) (Koordinaten bezüglich der Basis) und umgekehrt. Welche Basis wählt man? Die Wahl der Basis richtet sich nach dem Bereich, in dem die Werte a, b liegen sollen. Man geht so vor, daß man alle (c, d) aus einem Intervall $[c_0, \ldots, c_1] \times [d_0, \ldots, d_1]$ betrachtet. Dabei wählt man $d_0 = 1$,

$c_0 = -c_1$ und d_1 etwas kleiner als c_1. Die Anzahl aller zu untersuchenden Paare (c, d) bezeichnet man mit C. Dann gilt:

$$C = (2 \cdot c_1 + 1) \cdot d_1.$$

Um alle günstigen Paare (a, b) zu finden, geht man so vor: Man kann zeigen, daß wenn $\gcd(c, d) > 1$ ist, auch $\gcd(a, b) > 1$ ist.
Beweis:
Sei

$$(a, b) = c \cdot (a_1, b_1) + d \cdot (a_2, b_2),$$

wobei $g, c', d' \in \mathbb{Z}$ existieren, so daß

$$g = \gcd(c, d) > 1, c = c' \cdot g, d = d' \cdot g.$$

Dann gilt:

$$\begin{aligned} a &= g \cdot (c' \cdot a_1 + d' \cdot a_2) \\ b &= g \cdot (c' \cdot b_1 + d' \cdot b_2), \end{aligned}$$

so daß $\gcd(a, b) > 1$ ist.

Aber auch wenn $\gcd(c, d) = 1$ gilt, kann es vorkommen, daß

$$a \equiv b \equiv 0 \bmod q$$

ist. Diese wenigen Fälle werden in der Praxis nicht berücksichtigt. Um die erste Bedingung an (a, b) zu gewährleisten, bestimmt man diejenigen (c, d), für die $\gcd(c, d) = 1$ gilt.

Im zweiten Schritt soll gezeigt werden, wie man die zweite Bedingung an (a, b) verifiziert. Dieser Teil soll ABM-Sieb heißen, weil die Zerlegungseigenschaft von $a + bm$ untersucht wird.

Wie schon erwähnt, ist für alle $p < q$ zu entscheiden, ob $a + bm \equiv 0 \bmod p$. Dabei läßt sich die Kongruenz $a + bm \equiv 0 \bmod p$ umschreiben, indem man a, b durch ihre Darstellungen in c, d ersetzt.

$$\begin{aligned} (c \cdot a_1 + d \cdot a_2) + (c \cdot b_1 + d \cdot b_2) \cdot m &\equiv 0 \bmod p \\ c \cdot (a_1 + b_1 m) + d \cdot (a_2 + b_2 m) &\equiv 0 \bmod p \end{aligned}$$

Definiere dazu $u_1 = a_1 + b_1 m$ und $u_2 = a_2 + b_2 m$.
Dann ist folgende Kongruenz zu lösen:

$$c \cdot u_1 + d \cdot u_2 \equiv 0 \bmod p.$$

Man geht so vor, daß man für jedes Paar (c, d) eine Variable $V[c, d]$ einführt. Ziel ist es, für $V[c, d]$ folgenden Wert zu berechnen:

$$\sum_{p | a+bm} \log(p).$$

Dies erreicht man, indem man alle $p < q$ fest betrachtet und dann alle Paare (c, d) sucht, für die

$$c \cdot u_1 + d \cdot u_2 \equiv 0 \bmod p$$

gilt, und $V[c, d]$ um $\log(p)$ erhöht. Anschließend hängt für alle (c, d) von der Größe des Wertes $V[c, d]$ ab, ob das Paar noch weiter zu untersuchen ist, oder ob es sich nicht lohnt, es weiter zu betrachten. Hierbei heißt (c, d) ein günstiges Paar 2. Stufe, falls gilt:

- $\gcd(c, d) > 1$
- $V[c, d] + \mathrm{KORR} + \mathrm{MAXLARGE} > \log(a + bm)$

Dabei ist KORR eine Konstante, für die man ungefähr 5 wählt, die folgendes bewirkt:
In der Praxis werden die $\log(p)$ gerundet. So kann es vorkommen, daß der Endwert von $V(c, d)$ zu klein ist (kleiner als die gerundete Summe der Logarithmen). Außerdem spart man eine Menge Laufzeit, wenn man die Primzahlen bis zu einer gewissen Schranke (etwa die ersten 25) gar nicht berücksichtigt, so daß deren Logarithmen natürlich bei der Summe fehlen. Um dies zu korrigieren, wählt man diesen konstanten Wert KORR.

Der dritte Schritt geht fast genauso wie der zweite. Hier wird die dritte Bedingung, die sich mit der Zerlegungseigenschaft von $\mathrm{N}(a + b\rho)$ befaßt, untersucht. Deshalb heißt dieser Teil ABRHO-Sieb. Hier ist folgende Kongruenz zu verifizieren:

$$a + b \cdot cp \equiv 0 \bmod p.$$

Genauso wie im ABM-Sieb kann man dies durch Ersetzen von a, b durch ihre Terme in c, d umformen in

$$c \cdot u_1 + d \cdot u_2 \equiv 0 \bmod p,$$

wobei hier $u_1 = a_1 + b_1 \cdot cp$ und $u_2 = a_2 + b_2 \cdot cp$ definiert sind. Man geht wieder so vor, daß man für jedes Paar (c, d) eine Variable $V[c, d]$ einführt. Ziel ist es, für $V[c, d]$ folgenden Wert zu berechnen:

$$\sum_{p | a + b \cdot cp} \log(p).$$

Dies erreicht man, indem man alle (p, cp) fest betrachtet und dann alle Paare (c, d) sucht, für die

$$c \cdot u_1 + d \cdot u_2 \equiv 0 \bmod p$$

gilt, und $V[c, d]$ um $\log(p)$ erhöht.

Analog zum ABM-Sieb hängt für alle (c, d) von der Größe des Wertes $V[c, d]$ ab, ob das Paar noch weiter zu untersuchen ist, oder ob es sich nicht lohnt, es weiter zu betrachten.

Hierbei heißt (c, d) ein günstiges Paar 3. Stufe, falls gilt:

- (c, d) ist günstiges Paar 2. Stufe
- $V[c, d] + \mathrm{KORR} + 2 \cdot \mathrm{MAXPILARGE} > \log(\mathrm{N}(a + b\rho))$

Sowohl im ABM-Sieb als auch im ABRHO-Sieb ist folgendes Problem zu lösen. Sei p Primzahl und $0 < x < p$. Gesucht sind alle Paare $(c, d) \in [c_0, \ldots, c_1] \times [d_0, \ldots, d_1]$, für die gilt

$$c \equiv d \cdot x \bmod p.$$

Dieses Problem wird aus der Kongruenz

$$c \cdot u_1 + d \cdot u_2 \equiv 0 \bmod p$$

für den Fall, daß $u_1 \not\equiv 0 \bmod p$ ist, abgeleitet. Hier ist dann $x = u_2 \cdot u_1^{-1} \bmod p$. Der selten vorkommende, aber nicht zu vernachlässigende Fall, daß p Teiler von u_1 ist, soll zuerst abgehandelt werden. In diesem Fall kann man die Kongruenz zu

$$d \equiv 0 \bmod p$$

umformen. Man erkennt, daß die Lösungen unabhängig von c sind. Im konventionellen Sieb ist die Kongruenz

$$a + bm \equiv 0 \bmod p$$

für alle p zu lösen.
Hier hat man für jedes p sehr viele Paare (a, b), die dies erfüllen. Man hat sogar für jedes einzelne b viele a, so daß die Kongruenz erfüllt ist. Man betrachtet also alle Werte b nacheinander fest und bestimmt dann die zugehörigen a. Sei a_0 das kleinste a, das

$$a + bm \equiv 0 \bmod p$$

erfüllt. Dann läßt sich jedes a mit der geforderten Eigenschaft darstellen als

$$a = a_0 + k \cdot p,$$

wobei $0 \leq k$.

Mit dieser Methode kommt man bei dem Problem,

$$c \equiv x \cdot d \bmod p$$

zu lösen, nur bei kleinen Primzahlen p weiter.

Beispiel: Bei der Faktorisierung der Zahl $F_7 = 2^{2^7} + 1$ ergeben sich folgende Parameter:

- $p < 3571$
- $C = 12060$

- $c_1 = 100$
- $d_1 = 60$

Wieviele Lösungen existieren für ein festes d ? Da c genau $2c_1+1$ verschiedene Werte annehmen kann, erkennt man, daß für

- $p \leq 2c_1 + 1$ mindestens eine
- $p > 2c_1 + 1$ höchstens eine

Lösung existiert. Die Kongruenz hat also für die meisten Primzahlen wenige Lösungen (ungefähr C/p). Also kann das Verfahren aus dem konventionellen Sieb, $a+bm \equiv 0 \bmod p$ zu lösen, nicht angewandt werden, sondern muß verallgemeinert werden. Das Problem wird gelöst, indem ein Verfahren angegeben wird, das die Lösungen (c, d) so ermittelt: Wird (c_3, d_3) nach (c_2, d_2) berechnet, so ist $d_3 \geq d_2$.
Gesucht ist also ausgehend von $c = 0, d = 0$ immer die nächste Lösung (c, d) im folgenden Sinn: Es gibt keine weitere Lösung mit kleinerem d. Man geht so vor:
Zum Gitter aller Paare (c, d), die die Kongruenz

$$c \equiv x \cdot d \bmod p$$

erfüllen, berechnet man eine Gitterbasis, die aus den Vektoren (c_2, d_2) und (c_3, d_3) besteht. (c, d) werde durch die Basis so dargestellt:

$$(c, d) = e \cdot (c_2, d_2) + f \cdot (c_3, d_3).$$

An die Wahl der Gitterbasis werden Anforderungen gestellt. Es soll gelten:

- erhöht man e bei festem f, so wird d erhöht.
- für alle f gilt: es gibt ein e, so daß (c, d) Lösung ist.
- Für zwei Lösungen (c, d) mit
  $$(c, d) = e \cdot (c_2, d_2) + f \cdot (c_3, d_3)$$
  und (c^*, d^*) mit
  $$(c^*, d^*) = e^* \cdot (c_2, d_2) + f^* \cdot (c_3, d_3)$$
  gilt:
  $$f < f^* \Rightarrow d^* \geq d$$

Folgende Basis erfüllt diese Bedingungen nur für den Fall, daß $p < 2c_1 + 1$:

$$\{(p, 0), (x, 1)\}.$$

Für diesen Fall entspricht das hier beschriebene Verfahren dem Vorgehen im konventionellen Sieb. Falls man eine solche Basis gefunden hat, ergibt sich folgendes Vorgehen: Ist

$$(c, d) = e \cdot (c_2, d_2) + f \cdot (c_3, d_3)$$

eine Lösung, so findet man so die nächste:

- erhöhe e um 1
- Ist $|c| \leq c_1$, so gib (c, d) aus
- sonst erhöhe f um 1
- suche das kleinste e, so daß (c, d) Lösung ist und gib (c, d) aus. Wie dieses e zu berechnen ist, wird im folgenden Algorithmus deutlich.

Algorithmus:
Eingabe: $p \in P, x, c, d \in \mathbb{Z}$, wobei $d, x > 0$ und $c \equiv x \cdot d$
Ausgabe: das im obigen Sinn nächste Paar (c, d)

$(c_2, d_2) = (p, 0)$			(1)
$(c_3, d_3) = (x, 1)$			(2)
WHILE $c_2 > 2 \cdot c_1$			(3)
	$h = d_3$		(4)
	$d_3 = d_3 \cdot (c_2/c_3 + 1) - d_2$		(5)
	$d_2 = h$		(6)
	$h = c_3$		(7)
	$c_3 = c_3 - c_2 \bmod c_3$		(8)
	$c_2 = h$		(9)
$cd = -(c_3 + 2 \cdot c_1)/c_2$			(10)
$dd = cd \cdot d_2 + d_3$			(11)
$cd = cd \cdot c_2$			(12)
$cd = cd + c_3$			(13)
IF $c + c_2 > c_1$			(14)
THEN	$c = c + cd$		(15)
	$d = d + dd$		(16)
	IF $c < -c_1$		(17)
	THEN	$c = c + c_2$	(18)
		$d = d + d_2$	(19)
ELSE	$c = c + c_2$		(20)
	$d = d + d_2$		(21)

Bemerkungen:
In den Zeilen 1 und 2 wird eine Ausgangsbasis aus den Vektoren (c_2, d_2) und (c_3, d_3)

festgelegt. Daraus wird in der WHILE-Schleife eine Basis $\{(c_2, d_2), (c_3, d_3)\}$ derart bestimmt, daß durch einfaches Erhöhen der Koordinaten e und f das nächste Paar (c, d) erhalten wird. Der einfachste Fall taucht in Zeile 20 auf. Hier wird nur e um 1 erhöht, was bedeutet, daß c um c_2 und d um d_2 erhöht wird. Etwas schwieriger wird es, wenn der Fall eintritt, daß $c + c_2$ nicht mehr im vorgeschriebenen Bereich ist (Zeile 14). Dann wird f erhöht, was zur Folge hat, daß die in den Zeilen 10 bis 13 vorberechneten Werte cd und dd zu c und d aufaddiert werden. Ist man immer noch nicht im richtigen Bereich, so wird dies in Zeile 18 korrigiert.

2.2 Sieben in Runden

Jedem Paar (c, d) wird eine Variable $V(c, d)$ zugeordnet. Da bei größeren Zahlen N auch die Anzahl der zu untersuchenden Paare (c, d) groß wird, muß man aus Speicherplatzgründen den Siebbereich $[c_0, \ldots, c_1] \times [d_0, \ldots d_1]$ in Teilintervalle unterteilen, die dann nacheinander in mehreren Runden überprüft werden. Und zwar geht man folgendermaßen vor:
Sei INTERANZ die Anzahl der Runden. Dann betrachtet man die Intervalle $[c_0, \ldots, c_1] \times [1, \ldots, \text{INTERLEN}]$ bis $[c_0, \ldots, c_1] \times [d_1 - \text{INTERLEN} + 1, \ldots, d_1]$, wobei

$$\text{INTERLEN} = d_1 / \text{INTERANZ}$$

die Anzahl der verschieden d-Werte in einem Intervall sind. Als Speicherplatz wird nur noch $C/\text{INTERANZ}$ benötigt. Dabei taucht ein neues Problem auf, daß nun in jeder Runde für jedes p (bzw. (p, cp)) die Kongruenz

$$a + bm \equiv 0 \bmod p$$

in die Kongruenz

$$c \equiv x \cdot d \bmod p$$

umgewandelt werden muß. Diese Berechnungen nehmen sehr viel Zeit in Anpruch. Man führt also für jedes Faktorbasiselement eine Variable ein, in der man das entsprechende x abspeichert. Gleichzeitig führt man eine zweite Variable ein, in der man folgendes abspeichert:

- bei kleinen Primzahlen p den kleinsten gültigen c-Wert für den kleinsten d-Wert im Intervall
- bei großen Primzahlen das nächste gültige d

Dabei ist im zweiten Fall zu bedenken, daß es nicht möglich ist, für ein bestimmtes Intervall und ein $p \in FBR$ das erste gültige d in diesem Intervall zu bestimmen. Man kommt also hier um die zweite Variable nicht herum.

2.3 Realisierung des Lattice Sieve

Im folgenden soll beschrieben werden, wie man konkret vorgehen muß, um mit dem Lattice Sieve bei Benutzung eines Polynoms Paare (a, b) zu finden, die eine gute Chance haben, die geforderten Eigenschaften zu erfüllen. Hierbei geht man in mehreren Schritten vor.

- Man schreibt die speziellen Primzahlen, die man benutzen will, in eine Datei namens **datei**.
- Lese nächstes q aus **datei**.
- Berechne zu dem Gitter, das durch die Kongruenz

  $$a + bm \equiv 0 \bmod q$$

  definiert ist, eine geeignete Gitterbasis $\{(a_1, b_1), (a_2, b_2)\}$. Betrachte das Intervall $[c_0, \ldots, c_1] \times [1, \ldots, d_1]$. Jedem Paar (c, d) aus dem Intervall entspricht ein eindeutiges Paar (a, b) aus dem zu untersuchenden Siebintervall und umgekehrt.
- Vorberechnung
  - Durchlaufe die rationale Faktorbasis.
    Sei $p \in FBR$. Wandele die Kongruenz

    $$a + bm \equiv 0 \bmod p$$

    um in die Kongruenz

    $$c \equiv x_p \cdot d \bmod p$$

    und speichere den Wert x_p.
  - Durchlaufe die algebraische Faktorbasis.
    Sei $(p, cp) \in FBA$. Wandele die Kongruenz

    $$a + b \cdot cp \equiv 0 \bmod p$$

    um in die Kongruenz

    $$c \equiv x_{p,cp} \cdot d \bmod p$$

    und speichere den Wert $x_{p,cp}$.
- Durchlaufe in INTERANZ Runden die Teilintervalle $[c_0, \ldots, c_1] \times [d_2, \ldots, d_3]$, wobei $d_3 = d_2 + \text{INTERLEN}$.
  - Führe eine Variable $T[c, d]$ ein, wobei gilt:

    $$T[c,d] = \begin{cases} 1 & : \text{ falls } (a,b) \text{ nicht mehr brauchbar} \\ 0 & : \text{ sonst} \end{cases}$$

- Bestimme sämtliche (c,d) mit $\gcd(c,d) > 1$ und setze für sie $T[c,d]$ auf 1.
- Führe eine Variable $A[c,d]$ ein und setze $A[c,d]$ für alle c,d auf 0.
- Durchlaufe alle $p \in FBR$.
  Bestimme alle (c,d), für die gilt
  $$c \equiv x_p \cdot d \bmod p$$
  und erhöhe $A[c,d]$ um $\log(p)$.
- Auswertung
  Entscheide für alle (c,d), ob $A[c,d]$ groß genug. Falls nicht, setze $T[c,d]$ auf 1.
- Führe eine Variable $B[c,d]$ ein und setze $B[c,d]$ für alle c,d auf 0.
- Durchlaufe alle $(p,cp) \in FBA$.
  Bestimme alle (c,d), für die gilt
  $$c \equiv x_{p,cp} \cdot d \bmod p$$
  und erhöhe $B[c,d]$ um $\log(p)$.
- Auswertung
  Entscheide für alle (c,d), ob $B[c,d]$ groß genug. Falls nicht, setze $T[c,d]$ auf 1.

2.4 Bewertung des Algorithmus

Der Vorteil des Lattice Sieve Algorithmus gegenüber dem herkömmlichen Siebalgorithmus ist, daß viel weniger Paare (a,b) betrachtet werden. Als Nachteil ergibt sich, daß man nicht mehr alle günstigen Paare (a,b) findet. Es ist klar, daß diejenigen (a,b), für die $a+bm$ keinen Primteiler $q \in M$ hat, nicht erwischt werden. Man muß also soviele Primzahlen $q \in M$ betrachten, daß man einerseits wenige Paare (a,b) zu betrachten hat, andererseits aber nicht zuviele günstige Paare verschenkt. Dieses Problem soll im folgenden genauer untersucht werden. Hierzu werden nochmal die Mengen S und M definiert.

$$S = \{p \in \mathbb{P} \mid p \leq B_0\}$$

$$M = \{p \in \mathbb{P} \mid B_0 < p \leq B_1\}$$

Definiere zudem

$$k = \frac{B_0}{B_1}.$$

Zuerst soll berechnet werden, wieviele Paare (a,b) insgesamt betrachtet werden. Dabei ist es möglich, daß ein Paar (a,b) mehrfach betrachtet wird. Außerdem geht man davon aus, daß man für ein spezielles $q \in M$ A/q Paare (c,d) betrachtet, und nicht wie in der Praxis den Siebbereich für alle q gleich läßt. Hierbei ist $A \in \mathbb{N}$ eine Konstante, die angibt, wieviele Paare (a,b) insgesamt als günstige Paare in Frage kommen.

Man erhält insgesamt

$$A \cdot \sum_{q \in M} \frac{1}{q} = A \cdot \frac{\log \frac{1}{k}}{\log(B_1)}$$

zu überprüfende Paare (a,b). Jetzt soll untersucht werden, wieviele Paare (a,b) verloren werden. Zur Vereinfachung werden nur diejenigen Paare (a,b) betrachtet, für die $a + bm$ komplett über der rationalen Faktorbasis zerfällt. Im folgenden soll mit einem Beispiel von Pollard gezeigt werden, wie man 83% der Paare (a,b) mit 8% der gesamten Arbeit erhält. Betrachtet man Dickmann's ρ-Funktion (siehe [13]), so lassen sich Zerlegungseigenschaften natürlicher Zahlen als Werte dieser Funktion darstellen. Dann hat eine zufällig gewählte Zahl $a + bm$ nur Primfaktoren kleiner als B_1 mit einer Wahrscheinlichkeit

$$\rho(ln(bm)/ln(B_1)) \quad (3)$$

und nur Primfaktoren kleiner als B_0 mit einer Wahrscheinlichkeit

$$\rho(ln(bm)/ln(B_0)) \quad (4).$$

Im folgenden Beispiel wird die Approximationsformel

$$\rho(u) = exp(-u(ln(u) + 0.56 - 1/ln(u))), (5 < u < 11)$$

von Pomerance benutzt. In [13] steht genaueres darüber.

Das eben genannte Resultat läßt sich bei der Faktorisierung der Zahl $F_9 = 2^{2^9} + 1$ erreichen. Hier hat man als Werte $m = 2^{103}, b = 1250000, B_1 = 1300000$. In der folgenden Tabelle soll für verschiedene Werte von k mit W die zu verrichtende Arbeit und mit $L = (4)/(3)$ der Bruchteil der nicht berücksichtigten Paare (a,b) angegeben werden.

k	W	$ln(bm)/ln(B_0)$	(4)	$L = (4)/(3)$
1.0	0.0	6.0687	1.71(-5)	1.0
0.5	0.0492	6.3830	6.38(-6)	0.373
0.4	0.0651	6.4912	4.52(-6)	0.264
0.3	0.0855	6.6363	2.85(-6)	0.167
0.2	0.1143	6.8521	1.42(-6)	0.083
0.1	0.1636	7.2554	3.81(-7)	0.022

2.5 Laufzeit

Zur Vereinfachung wird die Laufzeit nur für den Algorithmus angegeben, der Paare (a, b) hinsichtlich der Zerlegung von $a + bm$ über der rationalen Faktorbasis untersucht, was der ersten Siebphase im Gesamtalgorithmus entspricht. Folgendes wird weiter vorausgesetzt:

- $\#q = \#M$ ist die Anzahl der verwendeten q -Werte
- für alle q wird ein Siebbereich der Größe
$$C = k \cdot A/B_1$$
gewählt, wobei k als Konstante ungefähr 10 ist und A die maximale Größe des Bereichs der Paare (a, b) angibt.

Die Laufzeit für ein spezielles q ergibt sich dann aus der Vorberechnung, dem Sieb und der Auswertung. Und zwar ergibt sich für die einzelnen Phasen

- In der Vorberechnung muß für alle $p < q$ die Kongruenz
$$a + bm \equiv 0 \bmod p$$
in die Kongruenz
$$c \equiv d \cdot x \bmod p$$
umgewandelt werden. Hierfür ergibt sich als Laufzeit
$$k_1 \cdot B_1'.$$
- In der Siebphase sind für jedes $p < q$ alle (c, d) zu bestimmen, für die
$$c \equiv d \cdot x \bmod p$$
gilt. Als Laufzeit für ein solches p nimmt man die Anzahl der Paare (c, d), die die Kongruenz erfüllen, also ungefähr
$$C/p.$$
Als Gesamtlaufzeit für diese Phase ergibt sich also
$$k_2 \cdot \sum_{p<q} C/p = k_2 \cdot C \cdot \sum_{p<q} 1/p.$$
- In der Auswertungsphase ist für jedes Paar (c, d) zu untersuchen, ob das Paar günstig ist, also durchläuft man alle Paare und erhält als Laufzeit
$$k_3 \cdot C.$$

k_1, k_2 und k_3 sind dabei Konstanten aus $\mathbb{N}$.
Als Gesamtlaufzeit des Algorithmus ergibt sich also
$$\#q \cdot (k_1 \cdot B_1' + k_2 \cdot C \cdot \sum_{p<q} 1/p + k_3 \cdot C).$$

2.6 Wahl der Polynome

Wie entscheidend die Wahl der Polynome für die Laufzeit des Algorithmus ist, zeigt sich im speziellen Number Field Sieve. Speziell bedeutet, daß die Koeffizienten sehr klein sind. In diesem Fall beträgt die Laufzeit nur $L[1/3, 1.5]$. Das hängt damit zusammen, daß die Größe der Norm entscheidend von den Koeffizienten des Polynoms abhängt und zwar gilt im Fall normierter Polynome:

$$\mathrm{N}(a + b\rho) = a^d - c_{d-1}a^{d-1}b + \ldots + (-1)^d c_0 b^d.$$

Im Fall nichtnormierter Polynome gilt:

$$\mathrm{N}(ac_d + b\omega) = (c_d a^d - c_{d-1}a^{d-1}b + \ldots + (-1)^d c_0 b^d) \cdot c_d^{d-1}.$$

Von der Wahl des Polynoms hängt auch die Wahl des Siebbereiches ab, also die Grenzen für die Paare (a, b). Davon wiederum hängt die Wahl der Gitterbasen ab.

Im folgenden wird für die Fälle

- spezielles Number Field Sieve
- Number Field Sieve mit normierten Polynomen
- Number Field Sieve mit nichtnormierten Polynomen

angegeben, wie der Siebbereich und die Gitterbasen zu wählen sind.

Fall 1: spezielles Number Field Sieve

Hier müssen die Maximalwerte für a und b etwa gleich groß sein. Grund: Die Koeffizienten des Polynoms $f(x)$ sind sehr klein. Um solche Paare (a, b) zu erreichen, muß eine Gitterbasis mit möglichst kleinen Vektoren gewählt werden.

Fall 2: Number Field Sieve mit normierten Polynomen

Hier gilt für den zweithöchsten Koeffizienten:

$$c_{d-1} < d.$$

Zur Erklärung dazu siehe [3] Seite 6. Der Maximalwert von a ist wesentlich größer als der von b zu wählen, was für den Fall $d = 3$ verdeutlicht werden soll. In diesem Fall gilt:

$$\mathrm{N}(a + b\rho) = a^3 - c_2a^2b + c_1ab^2 - c_0b^3$$

und die Größenordnung der Norm von $a + b\rho$ hängt von dem Summand c_1ab^2 ab.

Zu beachten ist auch, daß a nur bei der ersten Bedingung beliebig groß gewählt werden kann. Um solche Paare (a, b) zu erreichen, muß man geeignete Gitterbasen wählen, was sehr schwierig ist.

Fall 3: Number Field Sieve mit nichtnormierten Polynomen

Hier sind die Koeffizienten der Polynome alle von der gleichen Größenordnung, was zur Folge hat, daß auch die Maximalwerte von a und b gleich groß gewählt werden müssen. Also ist eine Gitterbasis mit möglichst kurzen Vektoren zu berechnen.

2.7 Gitter

Im Fall nichtnormierter Polynome ist es wichtig, für die Gitter Basen aus möglichst kurzen Vektoren zu bestimmen. Wie man aus der Beschreibung des Gitters zu einer solchen Basis gelangt, wird im folgenden erklärt.

Sei das Gitter
$$L = \{(a, b) \in \mathbb{Z}^2 \text{ mit } a + bm \equiv 0 \bmod q\}$$
gegeben. Gesucht ist eine Gitterbasis
$$\{(a_1, b_1), (a_2, b_2)\} \subset \mathbb{Z}^2 \times \mathbb{Z}^2$$
aus möglichst kurzen Vektoren.

Man geht so vor, daß man sich eine Ausgangsbasis betrachtet, nämlich
$$\{(-m, 1), (q - m, 1)\}.$$

Von ihr ausgehend berechnet man solange neue Basen, bis eine bestimmte Bedingung erfüllt ist. Zu folgendem Algorithmus siehe auch [12].

Algorithmus (Gitterbasisreduktion in der Ebene)
Eingabe: Gitter $L = \{(a, b) \in \mathbb{Z}^2 \mid a + bm \equiv 0 \bmod q\}$
Ausgabe: Gitterbasis $\{(a_1, b_1), (a_2, b_2)\}$

$(a_1, b_1) = (-m, 1)$			(1)
$(a_2, b_2) = (q - m, 1)$			(2)
$norm1 = a_1 \cdot b_1 + a_1 \cdot b_1$			(3)
$norm2 = a_2 \cdot b_2 + a_2 \cdot b_2$			(4)
IF $norm1 > norm2$			(5)
THEN	vertausche (a_1, b_1) und (a_2, b_2).		(6)
	$norm1 = norm2$		(7)
WHILE 1			(8)
	$skalar = a_1 \cdot b_1 + a_2 \cdot b_2$		(9)
	$qq = skalar/norm1$		(10)
	IF $qq < 1/2$		(11)
	THEN	break	(12)
	$(a_3, b_3) = (a_1, b_1) - q \cdot (a_2, b_2)$		(13)
	$norm3 = a_3 \cdot b_3 + a_3 \cdot b_3$		(14)
	IF $norm1 > norm3$		(15)
	THEN	$(a_2, b_2) = (a_1, b_1)$	(16)
		$(a_1, b_1) = (a_3, b_3)$	(17)
		$norm1 = norm3$	(18)
	ELSE	$(a_2, b_2) = (a_3, b_3)$	(19)

2.8 Der Parameter m

Bei der Besetzung der Parameter m_i bieten sich zwei Möglichkeiten an. Man probiert viele m_i aus und berechnet dazu passende Polynome f_i, so daß gilt:

$$f_i(m_i) \equiv 0 \bmod N.$$

Dabei ist man bemüht, Polynome mit möglichst guten Eigenschaften im Hinblick auf die Norm zu finden (zum Beispiel kleine Koeffizienten). Als andere Möglichkeit kann man nur ein m wählen, zu dem man dann Polynome berechnet, die

$$f_i(m) \equiv 0 \bmod N$$

erfüllen. Bei dieser Methode muß man zwar schlechtere (größere) Normwerte in Kauf nehmen, kann aber die Laufzeit der Siebphase und der Zerlegungsphase erheblich senken. Das hängt natürlich damit zusammen, daß die erste Bedingung dann für alle

Polynome gleich ist. Dies gilt bei mehreren m natürlich nicht, da dann ein Paar (a, b) zum Beispiel für alle Polynome die erste Bedingung erfüllen kann oder auch nur für ein Polynom, was von den voneinander unabhängigen Zerlegungen der $a + bm_i$ über der rationalen Faktorbasis FBR abhängt.

Kapitel 3

Die Zerlegung

Nach dem Sieb sind alle Paare (a, b) bestimmt, die eine gute Chance haben, die erforderlichen Eigenschaften zu erfüllen. Die Zerlegungsphase überprüft für sie jetzt, ob dies der Fall ist und berechnet dann die Vektoren, die dem Paar (a, b) zugeordnet sind. Dabei ist folgendes zu leisten:

- Es ist für jedes $p \in FBR$ zu überprüfen, ob p Teiler von $a+bm$ ist. So bestimmt man ein x, so daß
$$a + bm = x \cdot \prod_{i=0}^{B_1'-1} p_i^{e_i}.$$
Ist dieses x kleiner als die Schranke MAXLARGE, so erfüllt (a, b) die erste Bedingung und man hat
$$a + bm = lp(a, b) \cdot \prod_{i=0}^{B_1'-1} p_i^{e_i}.$$

- Es ist für jedes $(p, cp) \in FBA$ zu überprüfen, ob p Teiler von $a + b \cdot cp$ ist. So bestimmt man ein x, so daß
$$\mathrm{N}(a + b\rho) = x \cdot \prod_{i=0}^{B_2'-1} p_i^{e_i}.$$
Zerfällt dieses x in zwei Primfaktoren, die kleiner als die Schranke MAXPI-LARGE sind, so erfüllt (a, b) auch die zweite Bedingung und man hat
$$\mathrm{N}(a + b\rho) = lpi(a, b) \cdot \prod_{i=0}^{B_2'-1} p_i^{e_i}.$$

Man muß in der Zerlegung also genau das gleiche tun wie in der Siebphase mit dem Unterschied, daß man, wenn p Teiler von $a + bm$ ist, nicht den $\log(p)$ zu der dem

Paar (a,b) zugeordneten Variable addiert, sondern den Exponenten bestimmt und diese Information der Variable zuordnet. Analog verfährt man, wenn p Teiler von $\mathrm{N}(a+b\rho)$ ist.

Wenn man sehr wenige Paare (a,b) überprüfen muß (in der Double Large Prime Variante bleiben weniger übrig), kann man folgendermaßen vorgehen:

Man betrachtet jedes Paar (a,b) fest, durchläuft die rationale Faktorbasis und stellt fest, ob die erste Bedingung erfüllt ist. Falls ja, merkt man sich die Zerlegung von $a+bm$. Dann geht man die algebraische Faktorbasis durch und überprüft die zweite Bedingung. Ist auch diese erfüllt, so speichert man die zugehörigen Vektoren und die berechneten Largeprimes in einer Datei ab.

Hat man etwas mehr Paare (a,b) zu überprüfen (in der Triple Large Prime Variante), so geht man wie in der Siebphase vor.
Man überprüft zuerst die erste Bedingung für alle Paare (a,b). Hierbei durchläuft man die rationale Faktorbasis und berechnet für jedes $p \in FBR$ alle Paare (a,b), die

$$a + bm \equiv 0 \bmod p$$

erfüllen. Bei erfüllter Gleichung wird der Eintrag (p, e_p) in der (a,b) zugeordneten Variable vermerkt.

Hat man dies für alle p getan, so müssen diese Vermerke anschließend ausgewertet werden.

Für jedes (a,b) betrachtet man alle Vermerke (p, e_p) und multipliziert die p^{e_p} auf. Dann dividiert man $a+bm$ durch das Produkt und überprüft, ob das Ergebnis eine zulässige Largeprime ist.

Analog verfährt man bei der algebraischen Faktorbasis.

Kapitel 4

Berechnung der Teiler

Ist eine Lösung des linearen Gleichungssystems gefunden, so läßt sich daraus eine Kongruenz

$$x^2 \equiv y^2 \bmod N$$

konstruieren, die mit einer Wahrscheinlichkeit $> 1/2$ zu den Teilern der Zahl N führt. Dies soll begründet werden. Es wird gezeigt, daß $\gcd(x-y, N)$ ein echter Teiler von N ist, wenn gilt:

$$x \not\equiv \pm y \bmod N.$$

Also ist N Teiler von $x^2 - y^2 = (x+y) \cdot (x-y)$, aber N teilt weder $x+y$ noch $x-y$. Darum kann man N schreiben als

$$N = q_1 \cdot \ldots \cdot q_l \cdot \ldots \cdot q_{l+k}$$

mit $q_i \in \mathbb{P}$, $l \geq 1, k \geq 1$, so daß $x - y$ von $q_1 \cdot \ldots \cdot q_l$, nicht aber von $q_{l+1} \cdot \ldots q_{l+k}$ geteilt wird. Damit ist $\gcd(N, x-y) = q_1 \cdot \ldots \cdot q_l$.

Im folgenden soll die Realisierung dieses Abschlusses des gesamten Algorithmus erklärt werden. Man hat dabei diejenigen Mengen M_{ij} zu betrachten, für die x_{ij} in der Lösung 1 ist. Insgesamt läßt sich die Kongruenz gemäß der Herleitung aus folgenden Bausteinen konstruieren:

- die Polynome f_i und die zugehörigen g_i bzw. die Ableitungen.
- die Faktorbasis FBR

Algorithmus:

$x = 1$			(1)
$y = 1$			(2)
$e_k = 0$ für alle $0 \leq k < B_1'$			(3)
FOR alle Polynome f_i			(4)
	$n = 0$		(5)
	$\gamma = g_i'(\omega_i)^2$		(6)
	$x = x \cdot f_i'(m)$		(7)
	FOR alle benutzten Mengen M_{ij}		(8)
		$x = x \cdot lp(M_{ij})$	(9)
		$e_k = e_k + r_{ijk}$ für alle $0 \leq k < B_1'$	(10)
		$\gamma = \gamma \cdot \prod_{(a,b) \in M_{ij}} (ac_{id} + b\omega_i)$	(11)
		$n = n + \# M_{ij}$	(12)
	berechne β, so daß $\beta^2 = \gamma$		(13)
	$y = y \cdot \varphi_i(\beta)$		(14)
	$x = x \cdot c_{id}^{d-2+n/2}$		(15)
$x = x \cdot \prod_{0 \leq k < B_1'} p_k^{e_k/2}$			(16)
gib als Teiler $\gcd(x-y, N)$ und $\gcd(x+y, N)$ aus			(17)

Würde man so in der Praxis vorgehen, müßte man γ ausrechnen und dann die Wurzel von γ ziehen. Dies ist aber bei größeren Zahlen N nicht zu realisieren, weil man das Produkt von sehr vielen $ac_d + b\omega$ berechnen müßte. Dies geht nicht, weil die Ergebnisse und somit auch die Koeffizienten von γ viel zu groß sind. Außerdem ist es sehr schwer, die Wurzel aus einer algebraischen Zahl mit sehr großen Koeffizienten zu ziehen. Jetzt soll genau erklärt werden, wie man das Hauptproblem nach einer Idee von Couveignes ([6]) lösen kann. Dieses Vorgehen verlangt, daß der Grad d ungerade sein muß. Außerdem werden noch benötigt:

- die Vektoren a_{ijk}
- die algebraischen Faktorbasen FBA_i

Im folgenden wird beschrieben, wie man für ein Polynom $f = f_i$ vorgehen muß. Sei

$$\gamma = g'(\omega)^2 \cdot \prod_{(a,b) \in S} (ac_d + b\omega), \mathrm{N}(\gamma) = n^2, n \in \mathbb{N}$$

Ferner sei $\beta \in \mathbb{Z}[\omega]$ mit folgenden Eigenschaften:

$$\begin{aligned} \beta^2 &= \gamma \\ \mathrm{N}(\beta) &= n \end{aligned}$$

Ziel des Algorithmus ist es, $\varphi(\beta)$ zu bestimmen. Sei

$$b(x) = \sum_{i=0}^{d-1} b_i x^i \in \mathbb{Z}[x],$$

so daß

$$\beta = \sum_{i=0}^{d-1} b_i \omega^i,$$

dann gilt:

$$\varphi(\beta) \equiv b(c_d m) \bmod N.$$

Man muß also $b(c_d m) \bmod N$ berechnen. Da die b_i sehr groß sind, geht man so vor: Man betrachtet paarweise teilerfremde Moduln $m_j = p_j^{2^k}$ und berechnet $b(x) \bmod m_j$ und daraus $b(c_d m) \bmod m_j$. Dabei sind die p_j Primzahlen, für die gelten muß, daß $f(x) \bmod p_j$ irreduzibel ist. Der Wert k, die Anzahl der Moduln r und die Größe der Primzahlen p_j hängen von der Größe der Zahl N ab. Genaueres dazu steht in [6] auf Seite 4. Daraus läßt sich durch einfache Anwendung des chinesischen Restsatzes $b(c_d m) \bmod N$ ausrechnen. Problem: Wie berechnet man $b(x) \bmod m_j$?
Da $m_j = p_j^{2^k}$, berechnet man zuerst $b(x) \bmod p_j$ und liftet diese $b(x) \bmod p_j$ mit Hilfe der Newton-Iteration zu $b(x) \bmod m_j$ (siehe dazu [3] Seite 29).

Sei

$$c(x) = \sum_{i=0}^{d-1} c_i x^i \in \mathbb{Z}[x],$$

so daß

$$\gamma = \sum_{i=0}^{d-1} c_i \omega^i.$$

Da die Primzahlen p so gewählt sind, daß $\mathbb{Z}[\omega]/p\mathbb{Z}$ ein Körper aus p^d Elementen ist, läßt sich $b(x) \bmod p_j$ leicht berechnen.
Berechne eine Wurzel im Körper. Dies ist z.B. mit einer Variante von Shanks' RESSOL Algorithmus zu erreichen (siehe dazu [4]). So erhält man ein $w(x) \in \mathbb{Z}[x]$ mit

$$w(x)^2 \equiv c(x) \bmod (g(x), p_j).$$

Zur Bewertung dieses $w(x)$ wird folgende Definition eingeführt:

4.1. Definition *Sei $g(x) \in \mathbb{Z}[x]$ ein normiertes irreduzibles Polynom vom Grad d und $\omega \in \mathbb{C}$ eine komplexe Nullstelle von $g(x)$. Ferner sei $w(x) \in \mathbb{Z}[x]$ ein Polynom mit*

$$w(x) = \sum_{i=0}^{d-1} w_i x^i.$$

Dann definiere als $No(w(x), g(x))$ die Norm von $\sum_{i=0}^{d-1} w_i \omega^i$ im Ring $\mathbb{Z}[\omega]$.

Fall 1:

$$No(w(x), g(x)) \equiv n \bmod p_j$$

Dann gilt:

$$b(x) \bmod p_j = w(x).$$

Fall 2:

$$No(w(x), g(x)) \equiv -n \bmod p_j$$

Dann gilt:

$$b(x) \bmod p_j = -w(x).$$

Durch eine Erweiterung des chinesischen Restsatzes gelingt es,

$$\varphi(\beta) \bmod N$$

zu berechnen. Zuerst wird der chinesische Restsatz formuliert, dann wird gezeigt, wie man ihn auf dieses Problem anwenden kann.

4.2. Satz (Chinesischer Restsatz) *Seien m_i paarweise teilerfremde Moduln $\in \mathbb{Z}$, $x \in \mathbb{Z}$, u_i Zahlen $\in \mathbb{Z}$, wobei $0 < i \leq r$ und definiert man*

$$\begin{aligned} M &= \prod_{j=1}^{r} m_j \\ M_i &= M/m_i, \end{aligned}$$

so daß

$$x \equiv u_i \bmod m_i$$

gilt, so kann man über x aussagen:

$$x \equiv \sum_{i=1}^{r} u_i \cdot M_i \cdot (M_i^{-1} \bmod m_i) \bmod M.$$

Zum Beweis siehe [7] Seite 267.

Unter folgender Bedingung ist x durch die simultane Kongruenz eindeutig bestimmt:

$$-M/2 < x < M/2.$$

Denn dann gilt: x ist der absolut kleinste Wert, der

$$x \equiv u_i \bmod m_i$$

erfüllt.

In unserem Fall hat man $x = b(c_d m)$ modulo der Moduln m_j. Man ist jedoch nicht an der Zahl $b(c_d m)$ interessiert, sondern nur an ihrem Rest modulo N, da die entscheidende Kongruenz gilt:

$$\varphi(\beta) \equiv b(c_d m) \bmod N.$$

Man muß also den chinesischen Restsatz so erweitern, daß man $x \bmod N$ durch eine Formel ausrechnen kann. Dies geschieht in folgendem Satz:

4.3. Satz *Seien die Bedingungen des chinesischen Restsatzes gegeben, $N \in \mathbb{N}$. Sei weiterhin*

$$-M/2 < x < M/2$$

erfüllt, so daß x eindeutig bestimmt ist, so gilt für $x \bmod N$, wenn man für R diejenige ganze Zahl mit minimalem Abstand zu

$$\sum_{i=1}^{r} \frac{u_i \cdot (M_i^{-1} \bmod m_i)}{m_i}$$

setzt:

$$x \equiv \sum_{i=1}^{r} u_i \cdot M_i \cdot (M_i^{-1} \bmod m_i) - M \cdot R \bmod N.$$

Im folgenden soll gezeigt werden, wie man das Problem des Wurzelziehens einfacher lösen kann. Dies soll für normierte Polynome geschehen. Hierbei braucht man nicht mehr die Wurzel der großen Zahl γ zu betrachten, sondern kann stattdessen eine Zahl mit kleinerer Norm und damit kleineren Koeffizienten betrachten. Man geht dabei so vor:
In der Datei **SMALL_nfs** stehen Paare (a, b), die über der algebraischen Faktorbasis zerfallen, mit ihrer Zerlegung. Es wird eine Menge Q dieser Paare (a, b) gesucht, so daß γ durch $\prod_{(a,b)\in Q} (a + b\rho)^2$ in $\mathbb{Z}[\rho]$ teilbar ist.

Definiere dazu:

$$a_i(a, b) = e_{p_i, cp_i}(a, b).$$

$(a + b\rho)^2$ teilt γ nämlich in $\mathbb{Z}[\rho]$, wenn

$$2 \cdot a_i(a, b) \leq \sum_{(a,b)\in S} a_i(a, b) := A_i$$

für alle i.

Der dazugehörende einfache Algorithmus liefert folgende Ergebnisse:

$$\begin{aligned} x &= \prod_{(a,b)\in Q} (a + bm) \\ \alpha &= \prod_{(a,b)\in Q} (a + b\rho) \end{aligned}$$

$$a = \text{Vektor zu } \frac{\gamma}{\prod_{(a,b)\in Q} (a + b\rho)^2}$$

Für a gilt:

$$a_i = A_i - \sum_{(a,b)\in Q} 2 \cdot a_i(a, b).$$

Algorithmus:

- $\alpha = 1$
- $x = 1$
- $a_i := A_i = \sum_{(a,b)\in S} a_i(a,b)$
- Lese alle Paare (a, b) und zugehörige Info ein.
  Wenn $2a_i(a, b) \leq a_i$ für alle i

  1. $x = x \cdot (a + bm)$
  2. $a_i = a_i - 2a_i(a, b)$
  3. $\alpha = \alpha \cdot (a + b\rho)$

Kapitel 5

Implementierung

5.1 Die Bibliotheken libI und libF

Als Programmiersprache für die Implementierung wurde C++ gewählt. Diese Sprache stellt den Datentyp long zur Verfügung, der sich dazu eignet, einige Objekte des Algorithmus darzustellen. Als Beispiele sind der Grad D oder die benutzten Primzahlen zu nennen, also relativ kleine ganze Zahlen. Für die vielen sehr großen Zahlen, die sich als Resultate des Algorithmus ergeben, ist aber eine Langzahlarithmetik notwendig. Als Beispiele, wo die Zahlen viel größer als durch long darstellbare Zahlen sind, kann man die zu faktorisierende Zahl N und die Moduln $m_j = p_j^{2^k}$ anführen. In der Bibliothek **libI** wird eine Klasse Int zur Verfügung gestellt, die beliebig große ganze Zahlen und alle wichtigen Funktionen und Operatoren mit ihnen implementiert. Als zugehöriges Headerfile wird **Int.h** eingebunden.

Die Berechnung der Wurzel nach Couveignes, die in Kapitel 5 geschildert ist, macht es notwendig, rationale Zahlen darzustellen und mit ihnen zu rechnen (addieren und runden). Dazu wird die Bibliothek **libF** benutzt. Sie implementiert diese Zahlen und die Operatoren darauf. Das zugehörige Headerfile heißt **bigfloat.h**. Siehe zur Beschreibung die Diplomarbeit von Thomas Papanikolaou ([11]).

5.2 Headerfiles

Als Headerfiles werden die Dateien **global.h**, **extern.h**, **nfs.h** und **prot.h** benötigt. Dabei enthalten **global.h** und **extern.h** folgende globale Variablen:

- CHAR *lgsiebfeld .... Feld für Siebeinträge
- CHAR *PRIM .... Feld für rationale Faktorbasis

- long AKTP .... aktueller Index für p
- CHAR *P .... Feld für p von (p, cp)
- long *CP .... Feld für cp von (p, cp)
- long AKTPCP .... aktueller Index für (p, cp)
- Int N .... zu faktorisierende Zahl
- Int m .... der Wert m
- long *mmodp .... Feld für m mod p
- CHAR *primlog .... Feld für Logarithmen von p
- CHAR *plog .... Feld für Logarithmen von (p, cp)
- long START .... ab welcher Primzahl im ABM-Sieb
- long START1 .... ab welchem Paar (p, cp) im ABRHO-Sieb
- long Prime_number .... spezieller Index
- long P_number .... spezieller Index
- char KORR .... Konstante für Auswertung
- long D .... Grad
- long POLYANZ .... Anzahl der benutzten Polynome
- RING f .... aktuelles Polynom f
- long B .... Größe der rationalen Faktorbasis
- long ANZPI .... Größe der algebraischen Faktorbasis
- long MAXPI .... größtes Element der algebraischen Faktorbasis
- long MAXPRIM .... größtes Element der rationalen Faktorbasis
- long MAXLARGE .... größte zugelassene Largeprime
- long MAXPILARGE .... größte zugelassene algebraische Largeprime
- long MIN_C .... kleinster c-Wert
- long MAX_C .... größter c-Wert
- long MAX_D .... größter d-Wert
- long INTERANZ .... Anzahl der Runden im Sieb
- long INTERLEN .... Anzahl der d-Werte pro Runde

- long ADLEMAN_NR .... Größe der quadratischen Faktorbasis
- long **ADLEMAN .... hier wird die quadratische Faktorbasis gespeichert

Als Beispiel ist die globale Variable B in der Datei **global.h** als long B deklariert und in der Datei **extern.h** als extern long B. Also wird **global.h** in Hauptprogramme, die globale Variablen benutzen, eingebunden und **extern.h** in Dateien, die Funktionen implementieren.

In der Datei **prot.h** stehen die Prototypen der beschriebenen Funktionen.
In der Datei **nfs.h** sind als Abkürzungen definiert:

- ONE 1
- ZERO 0
- elba xgcd
- typedef Int* RING
- typedef long* Ring
- typedef unsigned char CHAR

5.3 Funktionen

Hier werden die Funktionen, die in der Implementierung benutzt werden, aufgeführt. Einige davon werden in der Diplomarbeit von Jürgen Loho ([8]) ausführlich beschrieben.

5.3.1 Funktionen mit ganzen Zahlen

- long invmodp(long x, long p)

  Die Funktion berechnet das Inverse von x modulo p, also eine Zahl y mit

  $$x \cdot y \equiv 1 \bmod p \quad \text{und} \quad 0 < y < p.$$

- long ggt(long a, long b)

  berechnet zu a und b den $\gcd(a, b)$

- BOOLEAN listprim(Int n)

  liefert 1, falls n Pseudoprimzahl und ansonsten 0.

- Int Idouble(double d)

  wandelt double d durch Runden in Int um.

- double llog(Int a)

  liefert den Logarithmus zur Basis e als double zurück.

- longasI(Int a)

  wandelt Int a in long um.

- void lpotmod(Int x, Int n, Int m, Int &erg)

  berechnet x^n modulo m und liefert das Ergebnis in erg.

- Int potenz(Int x, long n)

  liefert x^n als Ergebnis zurück.

- Int potenz_mod_n(Int x, long n, Int m)

  wie lpotmod, nur wird das Ergebnis zurückgegeben.

- long legendre(long a, long p)

  Die Funktion berechnet das Legendre-Symbol einer Zahl a und einer Primzahl p.
  Die Funktion liefert:

  $$\begin{array}{rl} 1, & \text{falls } a \text{ quadratischer Rest modulo } p \text{ ist} \\ 0, & \text{falls } \gcd(a,p) > 1 \\ -1, & \text{falls } a \text{ quadratischer Nichtrest modulo } p \text{ ist} \end{array}$$

- int listprim(Int n, long anzahl)

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

  $$\begin{array}{rl} 1, & \text{falls } n \text{ Pseudoprimzahl ist} \\ 0, & \text{falls } n \text{ als zusammengesetzt erkannt} \end{array}$$

- int squfof(Int N, long LP, long &p1, long &p2)

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

  $$\begin{array}{rl} 2, & \text{falls die gewünschte Zerlegung gefunden wurde} \\ 0, & \text{falls einer der Teiler zu groß ist} \\ 1, & \text{sonst} \end{array}$$

- int pollard(Int N, long LP, long &p1, long &p2)

  Die Funktion versucht, die beiden Teiler der 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

5.3.2 Einfache Hilfsfunktionen

- void phi(RING f, long d, Int &erg)

  berechnet für erg den Wert $f(m)$.

- Int pvonx(RING g, long d, Int x) liefert als Ergebnis $g(x)$ zurück, wobei g ein Polynom mit großen Koeffizienten aus $\mathbb{Z}$ vom Grad d ist.

- long abvonxmodp(Ring g, long x, long p)

  berechnet folgenden Wert: $g[0] + g[1] \cdot x \bmod p$.

- int gvonxmodp(Ring g, long x, long p, long d)

  g stellt ein Polynom g vom Grad d dar. Zurückgegeben wird $g(x) \bmod p$.

5.3.3 Funktionen zur Ringarithmetik

Für Polynome mit kleinen Koeffizienten aus $\mathbb{Z}$ (dargestellt als long) und Polynome mit großen Koeffizienten aus $\mathbb{Z}$ (dargestellt als Int) werden grundlegende arithmetische Funktionen zur Verfügung gestellt. Hier werden nur die Funktionen für letztere beschrieben, da die anderen analog sind (Ring statt RING, long statt Int). Dabei stelle ein Objekt f vom Typ RING ein Polynom f mit Koeffizienten aus $\mathbb{Z}$ dar, die durch den Datentyp Int dargestellt sind. Der zugehörige Grad df entspricht einer Variable df vom Typ long.

- R_add(RING f, RING g, Int a, RING h, long d)

  berechnet für h

  $$h = f + a \cdot g,$$

  wobei alle Polynome den maximalen Grad d haben.

- void R_Reduce(RING g, long *dg, RING f, long df)

  berechnet $g \bmod f$ und überschreibt g und dg mit dem Ergebnis.

- void R_mult_fak(RING g, long dg, Int a, RING f)

  berechnet $f = a \cdot g$.

- R_mult(RING f, long df, RING g, long dg, RING h, long dh, RING e, long *de)

  berechnet $e = f \cdot g \bmod h$.

- void RING_zuweisung(RING f, RING g, long d)

  setzt $f = g$, wobei der Grad d ist.

- void RING_neg(RING f, long df)

  negiert die Koeffizienten von f.

- int RING_vgl(RING f, RING g, long d)

  Der Grad ist d. Die Funktion liefert 1, falls $f = g$ und ansonsten 0.

- RING_add_l(RING f, RING g, Int a, RING h, long d, Int p)

  berechnet für h
  $$h = f + a \cdot g \bmod p,$$
  wobei alle Polynome den Grad d haben.

- void RING_Reduce(RING g, long *dg, RING f, long df, Int p)

  berechnet $g \bmod (f, p)$ und überschreibt g und dg mit dem Ergebnis.

- void RING_mult_fak(RING g, long dg, Int a, RING f, Int p)

  berechnet $f = a \cdot g \bmod p$.

- void RImulabp(RING ab, RING g, RING f, Int p, RING e)

  berechnet
  $$e = g \cdot (ab[0] + ab[1] \cdot x) \bmod f,$$
  wobei D der Grad von f ist.

- RING_mult(RING f, long df, RING g, long dg, RING h, long dh, RING e, long *de, Int p)

  berechnet $e = f \cdot g \bmod (h, p)$.

5.3.4 Funktionen für (p, cp)

Der Vollständigkeit halber werden die Funktionen, die bei der Berechnung der algebraischen und quadratischen Faktorbasen benötigt werden, aufgezählt:

- void make_x_pot_Liste(Ring f, long df, POLYNOM **Liste, long p)
- void mat_mult_vek(Ring *mat, Ring vek, long dim, long p, Ring erg)
- int pcp_make(long p, long *pcp, Ring f, long max_prim)

- void pascal(Ring *mat, long d, long a, long p)
- void invpas(Ring *mat, long d, Ring *erg)

5.3.5 Funktionen für die Faktorbasen

Die Faktorbasen müssen eingelesen und ihre Elemente nacheinander durchlaufen werden. Dazu stehen diverse Funktionen zur Verfügung.

- void p_diff_ein(long B)

  B ist die Größe der rationalen Faktorbasis. Ihre Elemente werden aus der Datei **diffprim** eingelesen.

- void pcp_diff_ein()

  Die algebraische Faktorbasis des gerade abgearbeiteten Polynoms wird eingelesen und zwar aus der Datei, deren Name in der globalen Variable DIFFPCP steht. Soll z.B. die algebraische Faktorbasis zum 2. Polynom eingelesen werden, dann muß DIFFPCP den Wert DIFF1 haben.

- void adle_diff_ein()

  Die quadratische Faktorbasis des gerade abgearbeiteten Polynoms wird eingelesen und zwar aus der Datei, deren Name in der globalen Variable ADLE steht. Soll z.B. die quadratische Faktorbasis zum 1. Polynom eingelesen werden, dann muß ADLE den Wert ADLE0 haben.

Die Funktionen

- int next_prime(long &p)
- int next_pcp(long &p, long &cp)
- int reset_prime(long &p)
- int reset_pcp(long &p, long &cp)

werden zum Durchgehen durch die Faktorbasen eingesetzt:

Will man die rationale Faktorbasis komplett durchlaufen, so geht man so vor:

- man setzt p auf 0.
- jeder Aufruf der Funktion next_prime(p) liefert die jeweils nächste Primzahl. Es wird 0 zurückgegeben, wenn die Faktorbasis durchlaufen ist.

Will man nur die Primzahlen durchlaufen, deren Logarithmen aufaddiert werden, so geht man so vor:

- man ruft auf: reset_prime(p)
- jeder Aufruf der Funktion next_prime(p) liefert die jeweils nächste Primzahl und es wird 0 zurückgegeben, wenn die Faktorbasis durchlaufen ist.

Um die algebraischen Faktorbasen komplett zu durchlaufen, geht man analog vor

- man setzt p auf 0.
- statt next_prime(p) ruft man next_pcp(p,cp) auf.

Um nur die Paare (p, cp), deren Logarithmen addiert werden, zu betrachten, geht man so vor:

- statt reset_prime(p) ruft man reset_pcp(p,cp)
- statt next_prime(p) ruft man next_pcp(p,cp) auf.

5.3.6 Funktionen zum Wurzelziehen

Folgende Funktionen werden beim Wurzelziehen benutzt:

- void alg_wurz(Ring quad, Ring mod, long D, Ring inv, long &d, long p)
  Die Parameter stellen folgende Objekte dar:
  - quad ein Polynom
    $$q(x) \in \mathbb{Z}[x] = \sum_{i=0}^{d-1} q_i x^i,$$
    so daß gilt:
    $$q_i \equiv c_i \bmod p,$$
    wobei für die c_i gilt:
    $$\gamma = \sum_{i=0}^{d-1} c_i \rho^i.$$
  - mod stellt das Polynom f modulo p dar.
  - inv stellt ein Polynom als Ergebnis der Funktion dar.

Die Funktion berechnet das Inverse der Wurzel von γ modulo p unter der Berücksichtigung, daß der Ring $\mathbb{Z}[x]/(p, f(x) \cdot \mathbb{Z}[x])$ ein Körper ist. inv stellt also am Ende ein Polynom

$$i(x) \in \mathbb{Z}[x] = \sum_{i=0}^{d-1} i_i x^i$$

dar, so daß gilt:

$$i(x)^2 \cdot c(x) \equiv 1 \bmod (p, f(x)),$$

wobei

$$c(x) = \sum_{i=0}^{d-1} c_i x^i.$$

- void newton_iteration(RING quad, RING wurzel, RING mod, long D, RING erg, long k, long p)

  Diese Funktion wird nach der Funktion alg_wurz aufgerufen. Hierbei stellt wurzel das Inverse der Wurzel von γ modulo p dar, entspricht also inv aus der Funktion alg_wurz. Sei $m = p^{2^k}$ der verwendete Modul. Weiterhin stellt quad γ modulo m und mod $f(x)$ modulo m dar. In erg wird die Wurzel von γ modulo m berechnet nach einer Newtoniteration. Dies entspricht einem Polynom $b(x)$ modulo m, wobei für $b(x) \in \mathbb{Z}[x]$ gilt:

  $$b(x) = \sum_{i=0}^{d-1} b_i x^i,$$

  so daß

  $$\beta = \sum_{i=0}^{d-1} b_i \rho^i$$

  diejenige Wurzel von γ mit Norm n ist.

- void primwurz_random(Ring f, long df, long p)

- void sucheprimwurz(long k, Int rest, Ring primwurz, Ring mod, long dmod, long p)

- make_inv(Ring element, Ring mod, long dmod, Ring inv, Int ord, long p)

5.3.7 Weitere wichtige Funktionen

- void teiler_abm(long *e, Int &erg)

  In e ist beim Aufruf folgender Vektor abgespeichert:

  $$e_k = \sum_{x_{ij}=1} r_{ijk}.$$

  Dann multipliziert die Funktion erg mit

  $$\prod_{k=0}^{B_1'-1} p_k^{e_k/2}.$$

- void teiler_abrho(long *e, RING norm, long n, RING pz)

  Die Variable n beinhaltet die Anzahl der bei der Anwendung des chinesischen Restsatzes verwendeten Primzahlen. Diese stehen in dem Feld pz. Weiterhin ist in e beim Aufruf folgender Vektor gespeichert:

  $$e_k = \sum_{x_{ij}=1} a_{ijk}.$$

  Dann multipliziert die Funktion für jeden der n Primzahlen $pz[j]$ den entsprechenden Wert der Norm $norm[j]$ mit

  $$\prod_{k=0}^{B_2'-1} p_{ik}^{e_k/2}.$$

- void next_d(long p, long x, long &c, long &d)

  Beim Aufruf müssen die Parameter die Kongruenz

  $$c \equiv x \cdot d \bmod p$$

  erfüllen. Die Funktion berechnet ein Paar (c, d), das dies auch erfüllt, wobei für d der nächste mögliche Wert gewählt wird.

- void gitter(long u1, long u2, long q, long &a1, long &b1, long &a2, long &b2)

  Zum Gitter aller Vektoren (c, d), die

  $$c \cdot u_1 + d \cdot u_2 \equiv 0 \bmod q$$

  erfüllen, wird eine Gitterbasis aus den möglichst kurzen Vektoren (a_1, b_1) und (a_2, b_2) berechnet.

- wurzelg(RING *q, RING norm, Int &y, RING, RING m, RING g, RING *g_pot)

  Die Funktion berechnet für das aktuell gültige Polynom f_i für diejenige Wurzel β_i von γ_i mit positiver Norm den Wert $\varphi_i(\beta_i)$. Er wird in y zurückgegeben.

- Int det_norm_modp(Ring g, Ring f, Int p)

  berechnet die Norm des durch g repräsentierten Elements im Ring $\mathbb{Z}[x]/f(x) \cdot \mathbb{Z}[x]$ modulo p.

- void fmodp(RING h, long dh, long p, Ring g)

  liefert in g das Polynom h modulo p zurück.

- void norm_abnull(long *ab, Int &erg)

  liefert in erg die Norm von $ab[0] + ab[1] \cdot \rho$ zurück.

- void poly_lesen()

  liest die verwendeten Polynome aus den Dateien, in denen sie abgespeichert sind, in eine globale Variable $g1$ ein.

5.3.8 Funktionen mit Polynomen

- void euklrest(Ring f, long &df, Ring g, long dg, long p)

  berechnet $f \bmod g$ und überschreibt f mit dem Ergebnis.

- int irredtest(Ring f, long df, long p)

  gibt 1 zurück, falls f modulo p irreduzibel ist und andernfalls 0.

- void polyinv(RING inv, long &di, RING f, long df, RING g, long dg, Int p)

  berechnet das Inverse zu f im Ring $\mathbb{Z}[x]/g \cdot \mathbb{Z}[x]$ und gibt es in inv und di zurück.

- void polyggt(Ring *f, long df, Ring *g, long *dg, long p)

  berechnet den GGT von f und g und überschreibt f mit dem Ergebnis.

- void reduce(Ring *f, long *df, Ring *g, long dg, long p)

  berechnet $f \bmod g$ und überschreibt f und df mit dem Ergebnis.

- void teile(Ring *f, long df, Ring *g, long *dg, long p)

  Die Funktion teilt f durch g modulo p. Das Ergebnis überschreibt g und dg.

5.4 Benutzung der Programme

In diesem Kapitel soll geschildert werden, wie man vorgehen muß, um eine beliebige Zahl N zu faktorisieren.

Als erstes muß man einige Parameter bestimmen und sie in Dateien schreiben. Danach müssen mehrere Programme nacheinander aufgerufen werden. Diese Programme lesen aus den Eingabefiles und produzieren ihrerseits wieder Ausgabefiles. Im folgenden sollen alle diese Schritte aufgelistet werden.

5.4.1 Berechnung der Faktorbasen

Durch den Parameter B, der im File **ein_nfs** steht, wird die Größe der rationalen Faktorbasis bestimmt. Die algebraischen und die quadratischen Faktorbasen müssen noch berechnet werden. Dies geschieht in dem Programm **pcp**. Dieses benötigt als Eingaben

- die Datei **ein_nfs**

  Aus ihr wird die Größe der algebraischen Faktorbasen ANZPI und die Größe der quadratischen Faktorbasen ADLEMAN_NR gelesen. Außerdem gibt die Variable POLYANZ die Anzahl der benutzten Polynome an.

- die Dateien **POLY0, POLY1,** ...

  In diesen Dateien stehen die benutzten Polynome.

- die Datei **primzahlen**

  Aus dieser Datei werden die Primzahlen p gelesen, für die nach Nullstellen cp der Polynome f_i modulo p gesucht wird, um die algebraischen Faktorbasen zu bestimmen.

- die Datei **BIGPRIMES**

  In dieser Datei stehen 9-stellige Primzahlen q. Diese werden eingelesen, um die Nullstellen cq der Polynome f_i modulo q zu suchen, die die quadratischen Faktorbasen bestimmen.

Das Programm **pcp** berechnet für jedes Polynom die algebraische und die quadratische Faktorbasis und gibt sie in folgenden Dateien aus:

- die Dateien **DIFF0, DIFF1,** ...

  Die algebraischen Faktorbasen.

- die Dateien **ADLE0, ADLE1,** ...

  Die quadratischen Faktorbasen.

5.4.2 Berechnung der Moduln

Für jedes Polynom ist eine Wurzel in dem dazugehörigen Zahlkörper zu ziehen. Dabei arbeitet man modulo mehrerer Primzahlen und setzt die Werte mit dem chinesischen Restsatz zusammen. Die Primzahlen p für das Polynom f müssen dabei als Bedingung erfüllen, daß f modulo p irreduzibel ist. Dabei sucht man für jedes Polynom die gleiche Anzahl solcher Primzahlen, die eine gewisse Größe haben sollen. Das Programm **inert** sorgt dafür. Es hat als Eingaben

- die Datei **ein_nfs**

  Aus ihr wird die Anzahl der benutzten Polynome POLYANZ und der Grad D gelesen.

- die Datei **primzahlen**

  Aus dieser Datei werden die in Frage kommenden Primzahlen gelesen.

- die Dateien **POLY0, POLY1,** ...

  Die Polynome müssen ja modulo der Primzahlen irreduzibel sein.

- Die untere Schranke und die Anzahl der zu berechneten Primzahlen für die Moduln ist einzugeben.

Das Programm **inert** liefert als Ausgabe die Dateien
INERT0, INERT1, ..., die die gewünschten Primzahlen erhalten.

5.4.3 Triviale Relationen

Triviale Relationen werden in dem Programm **trivial** berechnet. Es benötigt als Eingabefiles

- die Datei **ein_nfs**

  Daraus wird die Anzahl POLYANZ und der Grad D gelesen.

- die Files für die algebraischen Faktorbasen **DIFF0, DIFF1,** ...

  Die algebraischen Faktorbasen werden nach Primzahlen p abgesucht, zu denen es D Paare (p, cp) gibt.

- die Datei **diffprim**

  Die rationale Faktorbasis wird eingelesen, um ihr größtes Element zu bestimmen. Es werden nur die Primzahlen bis zu dieser Zahl genommen.

Da für die gefundenen Paare (a, b) gilt:

$$a = p \quad \text{und} \quad b = 0$$

und keine Largeprimes auftauchen, werden die zu (a, b) gehörigen Informationen in die Relationendateien angehängt. So entstehen als Ausgabe **rel0, rel1,** ....

5.4.4 Das Sieb

Das Programm **sieb** ist das Hauptprogramm des gesamten Programms. Es benutzt folgende Dateien:

- **ein_nfs**

  Alle in dieser Datei gespeicherten Parameter werden benötigt.

- die Dateien **POLY0, POLY1,** ...

- die Dateien **DIFF0, DIFF1,** ...

- die Dateien **ADLE0, ADLE1,** ...

- die Datei **diffprim**

sieb liefert als Ausgabedateien

- die Dateien **rel0, rel1,** ...
- die Dateien **lppi0, lppi1,** ...

Das Programm findet für jedes Polynom Paare (a, b), die gewisse Eigenschaften haben. Diese werden zusammen mit der Zerlegung der zugehörigen $a + bm_i$ bzw. $\mathrm{N}(a + b\rho)$ in den Ausgabedateien abgespeichert.

5.4.5 Kombinationen

Als Resultat des Siebes ergaben sich die Dateien **rel0, rel1,** ... und **lppi0, lppi1,** .... Die Paare (a, b), die in letzteren abgespeichert sind, müssen nun noch zu Mengen M_{ij} kombiniert werden. Dies geht für jedes Polynom gleich. Sei **lppi0** die Menge der Relationen für das erste Polynom.
1. Schritt:
Man ruft folgendes Programm auf:
zykzaehl lppi0 x MAXLARGE,
wobei x die Anzahl der Zeilen der Datei **lppi0** ist.
Als Ergebnis erhält man die Datei **lppi0.OUT**, in der alle Paare (a, b) abgespeichert sind, aus denen sich Mengen M_{0j} kombinieren lassen.
2. Schritt:
Man ruft folgendes Programm auf:
zykberech lppi0
Als Ergebnis erhält man die Datei **lppi0.ZYKLES**, in der steht, wie man die noch brauchbaren Paare (a, b) aus der Datei **lppi0.OUT** miteinander kombinieren muß. Diese beiden Programme werden für jedes Polynom aufgerufen. In Kapitel 6 sind einige Beispiele dazu aufgeführt.

5.4.6 das Programm makerels

Das Programm wird aufgerufen, nachdem folgende Voraussetzungen vorliegen: Die Mengen M_{ij} sind berechnet. Die Beschreibungen dazu sind im Fall einelementiger Mengen in den Dateien **rel0, rel1,** ... und im Fall mehrelementiger Mengen in den Dateien **lppi0.OUT, lppi1.OUT,** ... bzw. **lppi0.ZYKLES, lppi1.ZYKLES,** ... abgespeichert. Wie in der Herleitung beschrieben, gilt es nun, ein lineares Gleichungssystem zu lösen, das durch die den Mengen M_{ij} zugeordneten Vektoren r_{ij}, a_{ij} und q_{ij} bestimmt ist. Dieses Programm leistet folgendes:

- es leistet die Vorarbeit, um das Gleichungssystem lösen zu können, indem es die zugehörige Matrix berechnet und in einem gewissen Format in der Datei **RELATIONS** abspeichert. Diese Datei dient als Eingabedatei für die Programme, die das lineare Gleichungssystem lösen.

- Außerdem werden für jede Menge M_{ij} die Vektoren r_{ij} und a_{ij} berechnet und in der Datei **MATRIX** abgespeichert.

5.4.7 Lösen des LGS

Nachdem genügend Relationen gefunden wurden, gilt es nun, diejenigen zu suchen, die benötigt werden, um die Teiler der Zahl zu finden. Dies geschieht durch Lösen eines linearen Gleichungssystems. Die Matrix, die das Gleichungssystem bestimmt, wird durch die Datei **RELATIONS** beschrieben. Um Lösungen des Systems zu erhalten, sind nacheinander folgende Programme aufzurufen:

- **IGP**
- **exdense**
- **CFGauss**
- **exconvert** > **ALLSOLUTIONS**

Die Lösungen stehen danach in der Datei **ALLSOLUTIONS**.

5.4.8 Berechnung der Teiler

Nachdem das lineare Gleichungssystem gelöst ist, sind diejenigen Paare (a, b) bestimmt, die zu den Teilern der Zahl N führen. Um die entscheidende Kongruenz

$$x^2 \equiv y^2 \bmod N$$

zu finden, muß man im wesentlichen für jedes Polynom eine Wurzel einer sehr großen Zahl in einem Zahlkörper ziehen. Dies leistet das Programm **lgsg**.

Es benötigt als Dateien

- die Datei **ein_nfs**

  Hieraus liest es die Anzahl POLYANZ, den Grad D, die Zahl m.

- die Dateien **POLY0, POLY1,** ...

  Die Polynome definieren die Zahlkörper.

- die Datei **N**

  in ihr steht die zu faktorisierende Zahl N.

- die Datei **diffprim**

- die Dateien **DIFF0, DIFF1, ...**
- die Dateien **INERT0, INERT1, ...**

  die Moduln für den chinesischen Restsatz.
- die Datei **MATRIX**

  in ihr stehen die Informationen zu den Mengen M_{ij}.
- die Datei **ALLSOLUTIONS**

  In dieser Datei stehen Lösungen des Gleichungssystems, die alle mit Wahrscheinlichkeit $> 1/2$ zu den Teilern der Zahl führen. Und zwar stehen hier die Spaltenzahlen der Datei **MATRIX**, in der die Informationen zu einer Menge M_{ij} stehen, die an der Kongruenz beteiligt ist.

  Das Programm **lgsg** liest solange Lösungen aus **ALLSOLUTIONS**, bis nichttriviale Teiler gefunden werden. Diese werden ausgegeben.

5.5 Implementierung der Mehrkörpervariante

In diesem Abschnitt soll die Mehrkörpervariante mit dem Fall eines einzigen Polynoms verglichen werden. So soll herausgestellt werden, welche Teile des Programms parallelisiert werden können und wie der Ablauf sich bei der Benutzung mehrerer Polynome im Vergleich zum Number Field Sieve mit einem Polynom ändert. Dies wird für folgende Teile des Programms untersucht:

- Berechnung der algebraischen und quadratischen Faktorbasen
  Hier ist für alle Polynome das gleiche zu tun, nämlich die Paare (p, cp) zu bestimmen. So kann man auf einem Rechner die Faktorbasen des einen Polynoms bestimmen und auf einem anderen Rechner gleichzeitig ein anderes Polynom abhandeln.
- Das Sieb
  Wenn man die Parameter m_i nicht für alle Polynome gleich wählt, muß man für jedes Polynom das in Kapitel 2 beschriebene Lattice Sieve durchführen, was natürlich auch parallel auf mehreren Rechnern geschehen kann. In der Implementierung ist aber der Fall, daß man auch die m_i alle gleich wählen kann, bedacht worden. In diesem Fall läßt sich die Laufzeit erheblich verkürzen. Denn dann gilt: Die in Kapitel 3 genannten günstigen Paare (a, b) 2. Stufe, die
  - $\gcd(a, b) = 1$
  - $a + bm = lp \cdot \prod_{i=0}^{B_1^l - 1} p_i^{e_i}$

erfüllen, sind für alle Paare (a, b) gleich. Sie brauchen also nur einmal berechnet zu werden. Die Realisierung des Lattice Sieve mit mehreren Polynomen sieht folgendermaßen aus:

- Man schreibt die speziellen Primzahlen, die man benutzen will, in eine Datei namens **datei**.
- Lese nächstes q aus **datei**.
- Berechne zu dem Gitter, das durch die Kongruenz

  $$a + bm \equiv 0 \bmod q$$

  definiert ist, eine geeignete Gitterbasis $\{(a_1, b_1), (a_2, b_2)\}$. Betrachte das Intervall $[c_0, \ldots, c_1] \times [1, \ldots, d_1]$. Jedem Paar (c, d) aus dem Intervall entspricht ein eindeutiges Paar (a, b) aus dem zu untersuchenden Siebintervall und umgekehrt.
- Durchlaufe die rationale Faktorbasis.
  Sei $p \in FBR$. Wandele die Kongruenz

  $$a + bm \equiv 0 \bmod p$$

  um in die Kongruenz

  $$c \equiv x_p \cdot d \bmod p$$

  und speichere den Wert x_p.
- Durchlaufe in INTERANZ Runden die Teilintervalle $[c_0, \ldots, c_1] \times [d_2, \ldots, d_3]$, wobei $d_3 = d_2 + \text{INTERLEN}$.
  * Führe eine Variable $T[c, d]$ ein, wobei gilt:

    $$T[c, d] = \begin{cases} 1 & : \text{ falls } (a, b) \text{ nicht mehr brauchbar} \\ 0 & : \text{ sonst} \end{cases}$$

  * Bestimme sämtliche (c, d) mit $\gcd(c, d) > 1$ und setze für sie $T[c, d]$ auf 1.
  * Führe eine Variable $A[c, d]$ ein und setze $A[c, d]$ für alle c, d auf 0.
  * Durchlaufe alle $p \in FBR$.
    Bestimme alle (c, d), für die gilt

    $$c \equiv x_p \cdot d \bmod p$$

    und erhöhe $A[c, d]$ um $\log(p)$.
  * Auswertung
    Entscheide für alle (c, d), ob $A[c, d]$ groß genug. Falls nicht, setze $T[c, d]$ auf 1. Sonst speichere (c, d) in der Datei **Siebdaten** ab.
- Für alle Polynome f_i

  * Durchlaufe die algebraische Faktorbasis.
    Sei $(p, cp) \in FBA_i$. Wandele die Kongruenz

$$a + b \cdot cp \equiv 0 \bmod p$$

    um in die Kongruenz

$$c \equiv x_{p,cp} \cdot d \bmod p$$

    und speichere den Wert $x_{p,cp}$.
  * Durchlaufe in INTERANZ Runden die Teilintervalle $[c_0, \ldots, c_1] \times [d_2, \ldots, d_3]$, wobei $d_3 = d_2 + \text{INTERLEN}$.
    · Führe eine Variable $T[c, d]$ ein, wobei gilt:

$$T[c,d] = \begin{cases} 0 & : \text{ falls } (c,d) \text{ in } \textbf{Siebdaten} \text{ vermerkt} \\ 1 & : \text{ sonst} \end{cases}$$

    · Führe eine Variable $B[c, d]$ ein und setze $B[c, d]$ für alle c, d auf 0.
    · Durchlaufe alle $(p, cp) \in FBA_i$.
      Bestimme alle (c, d), für die gilt

$$c \equiv x_{p,cp} \cdot d \bmod p$$

      und erhöhe $B[c, d]$ um $\log(p)$.
    · Auswertung
      Entscheide für alle (c, d), ob $B[c, d]$ groß genug. Falls nicht, setze $T[c, d]$ auf 1.

- Kombinieren der Paare (a, b) zu Mengen M_{ij}
  Hier gilt das gleiche wie bei der Berechnung der Faktorbasen. Für jedes Polynom sind die gleichen Programme nacheinander aufzurufen.

- Lösen des linearen Gleichungssystems
  Hier hängt die Größe der Matrix des Gleichungssystems von der Anzahl der Polynome ab. Aus der Definition der Matrix in Kapitel 1 erkennt man auch, daß bei der Benutzung mehrerer Polynome die Matrix ein anderes Aussehen hat als bei der Benutzung eines einzigen Polynoms. Wenn man drei Polynome verwendet, sieht die Matrix so aus:

$$M = \begin{pmatrix} & A & \\ B & 0 & 0 \\ 0 & C & 0 \\ 0 & 0 & D \end{pmatrix}.$$

  Sei

$$S = \bigcup_{i=0}^{2} S_i.$$

  Dann gilt für die Teilmatrizen A, B, C, D:

  - A hat B_1' Zeilen und $\#S$ Spalten
  - B, C, D haben $B_2' + B_3'$ Zeilen
  - B hat $\#S_0$ Spalten
  - C hat $\#S_1$ Spalten
  - D hat $\#S_2$ Spalten

- Berechnen der Teiler
  Folgendes bezieht sich auf den Algorithmus in Kapitel 5 auf Seite 33. Es sei die Anzahl der Polynome zwei. Dann erhält man eine Kongruenz

  $$x^2 \equiv y^2 \cdot z^2 \bmod N.$$

  Dabei ergibt sich y aus der Wurzel einer algebraischen Zahl im durch das erste Polynom definierten Ring, z aus der Wurzel einer algebraischen Zahl im durch das zweite Polynom definierten Ring. Dabei hängen y und z jeweils nur von den Paaren (a, b), die ihrem Polynom zugeordnet sind, ab (y hängt von S_0 ab). Auf der anderen Seite hängt x von allen Paaren (a, b) ab.

5.6 Tricks

In diesem Kapitel werden Tricks aufgeführt, durch die sich die Laufzeit verbessern läßt, oder durch die Speicherplatz gespart werden kann.

5.6.1 Auswertungsphase

In der Auswertungsphase sind folgende Ungleichungen zu überprüfen:

$$\begin{aligned} lgs + \mathrm{KORR} + \mathrm{MAXLARGE} &> \log(a + bm) \\ lgs + \mathrm{KORR} + 2 \cdot \mathrm{MAXPILARGE} &> \log(\mathrm{N}(a + b\rho)) \end{aligned}$$

wobei lgs der zu dem Paar (a, b) gehörende Siebeintrag, KORR eine Konstante (ungefähr 5) und MAXLARGE bzw. MAXPILARGE die Grenzen für die Largeprimes sind. In der ersten Ungleichung spielt a keine Rolle. In der Praxis wird man nicht in jeder Abfrage einen Logarithmus ausrechnen, sondern beide Ungleichungen so umformen:

$$\begin{aligned} E[lgs + \mathrm{KORR} + \mathrm{MAXLARGE} - \log(m)] &> b \\ E[lgs + \mathrm{KORR} + 2 \cdot \mathrm{MAXPILARGE}] &> \mathrm{N}(a + b\rho) \end{aligned}$$

wobei E die Exponentialfunktion bedeutet. Diese Funktion kann man in einem Array für die in Frage kommenden Werte (0 bis 200) vorberechnen.

5.6.2 Die Logarithmen

Für jede Primzahl $p \in FBR$ muß $\log(p)$ und für jedes Paar $(p, cp) \in FBA$ muß $\log(p)$ berechnet werden. Diese Logarithmen kann man in Arrays abspeichern. Den dafür benötigten Speicherplatz kann man jedoch sparen, wenn man bedenkt, daß

- nur wenige Zahlen als Logarithmen vorkommen (ungefähr bis 200)
- wenn man von einem Faktorbasiselement zum nächsten geht, so wird der Logarithmus zuweilen (200 mal) erhöht.

Man kann also stattdessen so vorgehen, daß man in einem Array diejenigen Faktorbasiselemente (oder deren Indices) ablegt, bei denen sich der Logarithmus gegenüber dem Vorgänger um 1 erhöht.

5.6.3 Verdopplung der Spanne

Beim Bestimmen aller Paare (c, d), für die $a + bm \equiv 0 \bmod p$ bzw. $a + b \cdot cp \equiv 0 \bmod p$ gilt, kann man ausnutzen, daß nur solche (c, d) am Ende benutzt werden, die teilerfremd sind. Dabei beachte man insbesondere, daß für 1/4 aller Paare sowohl c als auch d gerade sind. Im folgenden Fall läßt sich, wenn man dies berücksichtigt, Laufzeit sparen.
Sei d gerade und fest, c_0 das kleinste ungerade c, für das $a + bm \equiv 0 \bmod p$ erfüllt ist. Dann sind alle gültigen c von der Form

$$c = c_0 + k \cdot p, \quad k = 0, 1, 2, \ldots$$

Hierbei stellt man fest, daß wenn k ungerade ist, c gerade wird. Würde man so alle c mit der Spanne p durchlaufen, so wäre jedes zweite c unbrauchbar, da der $\gcd(c, d) > 1$. Man kann also hier die Werte für c so durchlaufen, daß die Spanne zwischen zwei aufeinanderfolgenden Werten $2 \cdot p$ ist, daß sie also die Form

$$c = c_0 + k \cdot (2p), \quad k = 0, 1, 2, \ldots$$

haben.

5.7 Dateien

Vorab soll dargestellt werden, wie Vektoren $v = (v_0, \ldots, v_n)$, bei denen die meisten Komponeneten v_i gleich 0 sind, abgespeichert werden.

5.1. Definition *Zu einem Vektor $v = (v_0, \ldots, v_n)$ heißt die Aneinanderreihung von den Stellen i, an denen $v_i > 0$ ist, und v_i die kurze Darstellung von v. Mit $l(v)$ bezeichnet man die doppelte Anzahl der von 0 verschiedenen Werte v_i.*

Zum Beispiel ergibt sich für $v = (0, 0, 0, 2, 3, 0, 5)$ als kurze Darstellung 3 2 4 3 6 5.

1. **primzahlen**

   enthält die ersten 250000 Primzahlen.

2. **diffprim**

   enthält die Differenzen benachbarter Primzahlen, und zwar für die ersten 250000 Primzahlen.

3. **BIGPRIMES**

   enthält Primzahlen, die größer als MAXPILARGE sind. Aus ihnen werden die sogenannten Adlemaneinträge (q, cq) bestimmt.

4. **ein_nfs**

   enthält folgende Parameter des gesamten Algorithmus:

   - Anzahl der Polynome
   - Grad der Polynome
   - m
   - Größe der rationalen Faktorbasis B
   - Größe der algebraischen Faktorbasis ANZPI
   - Schranke für Largeprimes MAXLARGE
   - Schranke für Largeprimeideale MAXPILARGE
   - MAX_C
   - MAX_D
   - INTERANZ
   - INTERLEN
   - START
   - START1
   - KORR
   - ADLEMAN_NR

5. **INERT0, INERT1,** ...

   enthält für jedes Polynom Primzahlen, modulo derer f_i irreduzibel ist.

6. **DIFF0, DIFF1, ...**

   enthalten für jedes Polynom

   - ANZPI binär dargestellt
   - ANZPI mal für ein Paar (p, cp)
     - Differenz von p zur Primzahl p_1 des vorhergehenden Paares (p_1, cp_1) als char
     - cp binär dargestellt

7. **ADLE0, ADLE1, ...**

   Für jedes Polynom enthält die quadratische Faktorbasis B_3' Paare (q, cq). Diese sind in der Datei so abgespeichert, daß in einer Zeile q gefolgt vom zugehörigen cq steht.

8. **POLY0, POLY1, ...**

   enthalten die Koeffizienten der Polynome, und zwar stehen für $f(x) = c_d x^d + \ldots + c_0$ die $c_0, \ldots, c_d$ in dieser Reihenfolge jeweils in Stringdarstellung in einer Zeile.

9. **N**

   die zu faktorisierende Zahl N

10. **rel0, rel1, ...**

    Sei (a, b) ein zum Polynom f_i gehörendes Paar. Dann definiert man dazu folgende Vektoren:

$$\begin{aligned} r_k &= \mathrm{ord}_{p_k}(a + bm_i) \\ a_k &= e_{p_{ik}, cp_{ik}}(a, b) \\ q_k &= q_{q_{ik}, cq_{ik}}(a, b) \end{aligned}$$

    Zu (a, b) stehen folgende Informationen in einer Zeile:

    - 1
    - a
    - b
    - kurze Darstellung von r
    - -1
    - kurze Darstellung von a
    - -1
    - kurze Darstellung von q
    - -1

11. **lppi0, lppi1, ...**

Sei (a, b) ein zum Polynom f_i gehörendes Paar mit folgenden Eigenschaften:

$$\begin{aligned} a + bm_i &= p \cdot \prod_{k=0}^{B'_1-1} p_k^{r_k} \\ N(a,b) &= p_1 \cdot p_2 \cdot \prod_{k=0}^{B'_2-1} p_{ik}^{a_k} \end{aligned}$$

Falls dabei $p_i > 1$ für ein i ist, so seien cp_i die zugehörigen Nullstellen von $f_i(x)$. Die Vektoren r, a, q sind wie bei bei der Beschreibung der Dateien **rel0, rel1,** ... definiert. In Abhängigkeit von p, p_1, p_2 stehen folgende Informationen in einer Zeile:
Fall 1: $p = 1$
Fall 1.1: $p_i > 1, p_{3-i} = 1$

- 1
- p_i
- cp_i

Fall 1.2: $p_1 > 1, p_2 > 1$

- 2
- p_1
- cp_1
- p_2
- cp_2

Fall 2: $p > 1$
Fall 2.1: $p_i > 1, p_{3-i} = 1$

- 2
- p
- -1
- p_i
- cp_i

Fall 2.2: $p_1 > 1, p_2 > 1$

- 3
- p

- -1
- p_1
- cp_1
- p_2
- cp_2

Fall 2.3: $p_1 = 1, p_2 = 1$

- 1
- p
- -1

Danach kommen die gleichen Informationen wie bei den Dateien **rel0, rel1,** ....

12. **lppi0.OUT, lppi1.OUT,** ...

    enthalten diejenigen Zeilen aus den Dateien **lppi0, lppi1,** ..., die noch brauchbar sind.

13. **lppi0.ZYKLES, lppi1.ZYKLES,** ...

    verschiedene Paare (a, b) lassen sich zu einer Menge M kombinieren.
    In **lppi0.OUT, lppi1.OUT,** ... stehen die Informationen zu (a, b) in einer Zeile. Die Filepointer zu diesen Zeilen stehen nacheinander jeweils in einer Zeile dieser Dateien.

14. **MATRIX**

    Betrachte alle M_{ij}.
    Definiere folgenden Vektor v:

    - $v_0 = 0$
    - $v_{k+1} = r_{ijk}$ für $k = 0, .., B'_1 - 1$
    - $v_{k+1+B'_1} = a_{ijk}$ für $k = 0, .., B'_2 - 1$

    Sei $M_{ij} = \{(a_1, b_1), \ldots, (a_s, b_s)\}$, so stehen in der zugehörigen Zeile:

    - i
    - s
    - $a_1, b_1, \ldots, a_s, b_s$
    - $lp(M_{ij})$
    - $lpi(M_{ij})$, so daß

$$\mathrm{N}\left(\prod_{(a,b)\in M_{ij}} (a + b\rho_i)\right) = lpi(M_{ij})^2 \cdot \prod_{p\in FBA_i} p^{e_p}$$

- $l(v)$
- Darstellung von v wie die kurze Darstellung, aber mit dem Unterschied, daß zuerst der Wert und dann die Stelle kommt.

15. **RELATIONS**

    Sei (α_{ij}) die $m \times n$ Matrix, die dem linearen Gleichungssystem entspricht. In der j-ten Zeile der Datei **RELATIONS** stehen folgende Informationen zur j-ten Spalte der Matrix.

    Definiere den Vektor $v = (v_0, ..., v_m)$ durch $v_0 = 0, v_i = \alpha_{ij}$

    - k
    - $l(v)$
    - 0
    - Darstellung von v wie die kurze Darstellung, aber mit dem Unterschied, daß zuerst der Wert und dann die Stelle kommt.

16. **ALLSOLUTIONS**

    Die Lösungen des linearen Gleichungssystems stehen danach in der Datei **ALLSOLUTIONS** und zwar folgendermaßen:
    Die Datei besteht aus Zeilen, die alle aus einer Zahl bestehen. Die Zahl 7654321 trennt Lösungen voneinander. Ansonsten entsprechen den Zahlen der Datei die Spalten der Matrix, die in der Lösung vorkommen.

17. **SMALL_nfs0, SMALL_nfs1**, ...

    Diese Dateien entsprechen den Dateien **rel0, rel1**, ...

Kapitel 6

Praxis

6.1 Geeignete Parameter

Die Laufzeit des Algorithmus hängt im wesentlichen von folgenden Parametern ab:

- B_1' die Größe der rationalen Faktorbasis
- B_2' die Größe der algebraischen Faktorbasen
- E die Größe der Werte a, b, die der Wahl des Siebbereichs, also den Parametern c_0, c_1 und d_1 im Lattice Sieve entspricht.

Bei der Benutzung von n Polynomen müssen diese Parameter so gewählt werden, daß man etwa

$$\frac{B_1'}{n} + B_2'$$

Mengen M_{ij} pro Polynom erhält. Bei einem Polynom benötigt man also $B_1' + B_2'$ Mengen M_{ij} und muß deshalb die algebraische Faktorbasis wesentlich größer als die rationale wählen. Anhand von Beispielen soll gezeigt werden, welche Möglichkeiten der Parameterwahl sich bei der Benutzung mehrerer Polynome bieten. Man geht von folgendem Fall aus: Bei der Faktorisierung einer 29-stelligen Zahl benutzt man ein Polynom und wählt als Parameter

- $B_1' = 700$
- $B_2' = 2500$

Um die erforderlichen 3200 Mengen zu finden, muß man die Paare (c, d) aus einem Bereich $[-500, \ldots, 500] \times [1, \ldots, 50]$ betrachten. Es werden nun zwei Möglichkeiten gezeigt, wie man bei der Benutzung zweier Polynome verfahren kann.

- man kann die Größe der Faktorbasen gleich lassen. Dann braucht man pro Polynom nur noch
$$2850 = \frac{700}{2} + 2500$$
Mengen zu suchen. Um diese zu finden, braucht man jedoch nicht mehr das gleiche Siebintervall wie zuvor, sondern kann sich auf die kleineren Werte (c, d) beschränken.

- Man kann aber auch die rationale Faktorbasis verdoppeln, also
  - $B'_1 = 1400$
  - $B'_2 = 2500$

  wählen. Dann braucht man genauso wie im Fall eines Polynoms 3200 Mengen pro Polynom. Würde man für das Siebintervall wieder $[-500, \ldots, 500] \times [1, \ldots, 50]$ nehmen, so würde man aber viel mehr als 3200 pro Polynom erhalten, also kann man sich auch hier auf kleine Paare (c, d) beschränken.

Dabei ist zu beachten, daß man bei der Verdopplung des Siebintervalls viel weniger als doppelt so viele Mengen erhält. Ein weiteres Beispiel zeigt, wie man durch geschickte Parameterwahl mit zwei Polynomen die gesamte Laufzeit deutlich senken kann. Hierbei werden bei einer 14-stelligen Zahl zunächst bei einem Polynom folgende Parameter gewählt:

- $B'_1 = 50$
- $B'_2 = 200$

Um die erforderlichen 250 Mengen zu erhalten, muß das Siebintervall $[-200, \ldots, 200] \times [1, \ldots, 200]$ untersucht werden. Wenn man zwei Polynome verwendet und die rationale Faktorbasis verdoppelt, braucht man pro Polynom wiederum 250 Mengen M_{ij}. Um diese zu finden, kann man sich aber auf einen Siebbereich von $[-50, \ldots, 50] \times [1, \ldots, 25]$ beschränken, man muß also nur noch $1/32$ der ursprünglich untersuchten Paare (c, d) betrachten. Die gesamte Laufzeit konnte dadurch um den Faktor 8 verbessert werden.

6.2 Bewertung der Mehrkörpervariante

In diesem Abschnitt soll gezeigt werden, daß sich durch die Verwendung mehrerer Polynome zwar die theoretische Laufzeit verbessern läßt, aber dies sich nicht in die Praxis umsetzen läßt. Bei kleineren Zahlen (etwa 30 Dezimalstellen) läßt sich die Laufzeit zwar etwas verbessern, aber in diesem Bereich sind andere Faktorisierungsverfahren viel schneller. Zumindest bei größeren Zahlen N (etwa ab 40 Dezimalstellen) lassen sich keine günstigen Parameterkonstellationen mit der Benutzung mehrerer Polynome finden.

Es sollen drei Gründe genannt werden, warum das so ist.
Der erste Grund hängt mit zwei Punkten zusammen

- Die Werte $a+bm$ und $\mathrm{N}(a+b\rho)$ haben stark unterschiedliche Größenordnungen. Wenn die zu faktorisierende Zahl N 120 Dezimalstellen hat, so hat $a+bm$ 27 Dezimalstellen und $\mathrm{N}(a+b\rho)$ 55 Dezimalstellen. Bei der Benutzung eines Polynoms hat dies selbstverständlich zur Folge, daß die algebraische Faktorbasis wesentlich mehr Elemente hat als die rationale. So hat bei der Faktorisierung einer 49-stelligen Zahl die rationale Faktorbasis 3300 Elemente und die algebraische Faktorbasis 20000 Elemente.

- Die vier Parameter

  - Anzahl der Polynome n
  - Größe der rationalen Faktorbasis B_1'
  - Größe der algebraischen Faktorbasis B_2'
  - C, die Anzahl der Paare (c, d)

  hängen so miteinander zusammen, daß man $\frac{B_1'}{n} + B_2'$ Mengen M für jedes Polynom finden muß, wenn man C Paare (c, d) im Lattice Sieve untersucht. Wäre die Anzahl der erforderlichen Mengen $B_1' + \frac{B_2'}{n}$, so wäre das Problem gelöst.

Der Hauptgrund dafür, warum das Verfahren bei sehr großen Zahlen nicht praktikabel ist, hängt mit der Benutzung von Largeprimes zusammen. Die Paare (a, b), für die sowohl $a + bm$ über der rationalen Faktorbasis als auch $\mathrm{N}(a + b\rho)$ über der algebraischen Faktorbasis zerfallen, machen hier nur einen geringen Teil aus. Die restlichen Mengen M_{ij}, die aus mehreren (a, b) zusammenkombiniert werden, erhält man jedoch erst, wenn man einen großen Siebbereich für alle speziellen q-Werte im Lattice Sieve untersucht hat (siehe die Untersuchungen in [10]). Das bei der Benutzung mehrerer Polynome erforderliche Einschränken des Siebbereichs hätte zur Folge, daß man zwar nicht übermäßig viele Paare (a, b) mit Largeprimes verschenkt, aber aus diesen nicht genügend Kombinationen bilden kann.

Ein weiteres Problem bei der Benutzung mehrerer Polynome ergibt sich dadurch, daß die Größe des linearen Gleichungssystems von der Anzahl abhängt. Die zugehörige Matrix hat nämlich etwa $B_1' + n \cdot B_2'$ Zeilen und Spalten.

Zur Verdeutlichung soll noch ein erfolgloser Versuch, eine 40-stellige Zahl mithilfe zweier Polynome schneller als mit einem zu faktorisieren, dargestellt werden. Bei der Benutzung eines Polynoms wurden dabei folgende Parameter gewählt:

- $B_1' = 2000$
- $B_2' = 7500$
- $[c_0, \ldots, c_1] \times [d_0, \ldots, d_1] = [-500, \ldots, 500] \times [1, \ldots, 200]$

Nach 28 Stunden waren die erforderlichen 9500 Mengen gefunden. Bei der Benutzung zweier Polynome wurden die Größen der Faktorbasen gleich gelassen, was zur Folge hat, daß man pro Polynom 1000 Mengen weniger benötigt. Nun wurde versucht, mit einem kleineren Siebbereich zum Ziel zu gelangen. Dabei kam es zu folgenden Ergebnissen:

- Um die gleiche Laufzeit wie bei der Verwendung von einem Polynom zu erreichen, mußte der Siebbereich auf $[-120, \ldots, 120] \times [1, \ldots, 100]$ reduziert werden. Dabei wurden aber weniger als die Hälfte der erforderlichen 8500 Mengen pro Polynom erlangt.
- Bei einem Siebbereich von $[-400, \ldots, 400] \times [1, \ldots, 140]$ wurde pro Polynom das Ziel erreicht. Dabei betrug die Gesamtlaufzeit aber 40 Stunden.

6.3 Beispiele

Die Benutzung der Programme soll an einem Beispiel erklärt werden. Dabei wird eine 29-stellige Zahl mithilfe zweier Polynome faktorisiert. Die Zahl steht in der Datei **N**, die so aussieht:
68094773835969194533114212277

Die verwendeten Polynome werden in den Dateien **POLY0** und **POLY1** gespeichert. Sie sehen so aus:

- **POLY0**
  219569758
  -552450799
  1
  1

- **POLY1**
  219569758
  -552450799
  1
  1

Die wichtigsten Parameter stehen in der Datei **ein_nfs**.
2
3
4083550467
700
100000
2500
100000
500
50
1
50
20
20
3
10

Die Dateien, in denen die quadratischen Faktorbasen gespeichert sind, sehen so aus:

- **ADLE0**
  581534141 404625607
  ⋮
  605934949 546221354

- **ADLE1**
  581534141 491849645
  ⋮
  588502877 155618340

Die Dateien, in denen die Primzahlen, die als Moduln bei der Anwendung des chinesischen Restsatzes dienen, gespeichert sind, sehen so aus:

- **INERT0**
  30101
  ⋮
  38351

- **INERT1**
  30071
  ⋮
  36821

Die Paare (p, cp), aus denen die algebraischen Faktorbasen bestehen, sehen so aus:

- für **DIFF0**
  2 0
  7 6
  11 10
  $\vdots$
  22273 13120
  22277 19848
  22291 9801

- für **DIFF1**
  3 1
  5 1
  7 4
  $\vdots$
  22279 2263
  22279 17447
  22291 17574

In der Datei **datei** stehen alle Primzahlen zwischen 822 und 5279, wobei 822 die 100. und 5279 die 700. Primzahl ist. **datei** sieht also so aus:
822
$\vdots$
5279
0

Nachdem die Files **DIFF0**, **DIFF1** berechnet sind, startet man das Programm **sieb** und erhält als Ausgabedateien

- **rel0**, **rel1**
- **lppi0**, **lppi1**

Nachdem die Programme **zykzaehl** und **zykberech** aufgerufen sind, sind alle Mengen M_{ij} berechnet. Danach wird das Programm **makerels** aufgerufen. Es speichert die zu den Mengen M_{ij} gehörenden Informationen in der Datei **MATRIX** und stellt die Datei **RELATIONS** auf, die die Beschreibung des linearen Gleichungssystems enthält. Im folgenden sollen die Informationen zu den Mengen M_{ij} für beide Polynome $(i = 1, 2)$ sowohl für einelementige als auch für kombinierte Mengen M_{ij} aufgelistet werden. Die zugehörigen Vektoren sind in **rel0, rel1** bzw. in **lppi0.OUT, lppi1.OUT** abgespeichert.

In der Datei **MATRIX** steht die gleiche Information, jedoch in etwas anderer Form. Aus der Datei **RELATIONS** soll der zu x_{ij} gehörende Eintrag, der die entsprechende Spalte der Matrix beschreibt, dargestellt werden.

- **rel0**
  1 29505 38 1 1 8 1 12 1 34 1 100 1 121 1 -1 3 1 62 1 87 1 199 1 281 1 664 1 -1 0 1 1 1 5 1 7 1 -1 1
- **MATRIX**
  0 1 29595 38 1 1 26 1 2 1 9 1 13 1 35 1 101 1 122 1 704 1 763 1 788 1 900 1 982 1 1365
- **RELATIONS**
  4643 34 0 1 2 1 9 1 13 1 35 1 101 1 122 1 704 1 763 1 788 1 900 1 982 1 1365 1 3194 1 3195 1 3199 1 3201 0

- **rel1**
  1 -2461 16 47 1 49 1 100 1 345 1 -1 0 1 1 1 4 1 11 1 110 1 790 1 1607 1 -1 0 1 1 1 8 1 -1 1
- **MATRIX**
  1 1 -2461 16 1 1 22 1 48 1 50 1 101 1 346 1 701 1 702 1 705 1 712 1 811 1 1491 1 2308
- **RELATIONS**
  18559 28 0 1 48 1 50 1 101 1 346 1 3204 1 3205 1 3208 1 3215 1 3314 1 3994 1 4811 1 5738 1 5739 1 5746 0

- **lppi0.OUT**
  1 10009 -1 1 240182 7 99 1 699 1 -1 0 5 120 1 155 1 2115 1 2290 1 -1 0 1 2 1 3 1 5 1 -1 1
  1 10009 -1 1 263591 192 2 1 4 1 101 1 374 1 -1 297 1 440 1 1051 1 1095 1 1444 1 -1 0 1 1 1 3 1 4 1 5 1 8 1 -1 1

- **MATRIX**
  0 2 263591 192 240182 7 10009 1 32 1 3 1 5 1 100 1 102 1 375 1 700 5 701 1 821 1 856 1 998 1 1141 1 1752 1 1796 1 2145 1 2816 1 2991

- **RELATIONS**
  1 44 0 1 3 1 5 1 100 1 102 1 375 1 700 5 701 821 1 856 1 998 1 1141 1 1752 1 1796 1 2145 1 2816 1 2991 1 3194 1 3195 1 3197 1 3195 1 3198 1 3202

- **lppi1.OUT**
  1 10039 -1 1 -108830 9 213 1 407 1 -1 2 1 17 1 25 1 67 1 119 1 127 1 206 1 -1 1 1 2 1 3 1 4 1 8 1 -1 1
  1 10039 -1 1 -18779 271 0 1 16 1 158 1 167 1 -1 3 2 87 1 122 1 181 1 368 1 1537 1 -1 3 1 4 1 5 1 7 1 8 1 9 1 -1 1

- **MATRIX**
  1 2 -108830 9 -18779 271 10039 1 48 1 1 1 17 1 159 1 168 1 214 1 408 1 703 2 704 1 718 1 726 1 768 1 788 1 820 1 828 1 882 1 907 1 1069 1

- **RELATIONS**
  10338 48 0 1 1 1 17 1 159 1 168 1 214 1 408 1 3206 2 3207 1 3221 1 3229 1 3271 1 3291 1 3323 1 3331 1 3385 1 3410 1 3572 1 5739 1 5740 1 5743 1 5745 1 5747

Für das erste Polynom soll als Beispiel angegeben werden, wie man aus sechs Paaren (a, b) eine Menge M_{1j} kombinieren kann.

- **lppi0.OUT**
  2 99527 -1 47869 8751 1 43685 88 259 1 326 1 -1 27 1 43 1 66 1 83 1 205 1 -1 3 1 5 1 7 1 8 1 9 1 -1 1
  2 99527 -1 56767 54515 1 -16013 371 0 2 183 1 486 1 -1 3 1 554 1 1697 1 2399 1 -1 0 1 1 1 3 1 8 1 9 1 -1 1
  2 31583 -1 47869 8751 1 -5440 389 4 1 6 1 25 1 383 1 -1 0 1 3 1 129 1 431 1 1662 1 -1 0 1 2 1 4 1 5 1 8 1 9 1 -1 1
  1 31583 -1 1 1286 477 2 1 21 1 43 1 139 1 -1 0 2 3 1 6 2 22 1 255 1 356 1 947 1 -1 4 1 7 1 9 1 -1 1
  2 12289 -1 56767 54515 1 378720 169 1 1 7 1 72 1 586 1 -1 0 1 21 1 25 1 61 1 301 1 2018 1 2393 1 -1 1 1 3 1 4 1 6 1 7 1 8 1 9 1 -1 1
  1 12289 -1 1 33046 9 135 1 538 1 -1 0 2 23 1 66 1 153 1 205 1 1248 1 -1 2 1 5 1 -1 1

- **MATRIX**
  0 5 43685 88 -16013 371 -5440 389 1286 477 378720 169 33046 9 1 1 1 2 1 1 2 1 3 1 5 1 7 1 8 1 22 1 26 1 44 1 73 1 140 1 136 1 184 260 1 327 1 384 1 487 1 539 1 587 8 701 3 704 2 707 1 723 1 726 1 728 1 744 1 762 2 767 1 784 1 830 1 854 1 884 1 956 2 906 1 1002 1 1057 1 1132 1 1187 1 1648 1 1949 1 2363 1 2719

- **RELATIONS**
  1 40 0 9 1 1 1 2 1 1 2 1 3 1 5 1 7 1 8 1 22 1 26 1 44 1 73 1 140 1 136 1 184 260 1 327 1 384 1 487 1 539 1 587 8 701 3 704 2 707 1 723 1 726 1 728 1 744 1 762 2 767 1 784 1 830 1 854 1 884 1 956 2 906 1 1002 1 1057 1 1132 1 1187 1 1648 1 1949 1 2363 1 2719 1 3197 1 3198 1 3199 1 3200 1 3201 1 3203

Literaturverzeichnis

[1] L. M. Adleman, *Factoring numbers using singular integers*, Proc. 23rd Annual ACM Symp. on Theory of Computation (STOC), New Orleans, May 6-8, 1991, 64-71

[2] D. J. Bernstein, A. K. Lenstra, *A general number field sieve implementation*, Preprint 1992

[3] J. P. Buhler, H.W. Lenstra, Carl Pomerance, *Factoring integers with the number field sieve*, Preprint 1992

[4] J. Buchmann, J. Loho, J. Zayer, *An implementation of the general number field sieve*, Manuskript, 1993

[5] D. Coppersmith, *Modifications to the number field sieve*, IBM Research Report #RC 16264, Yorktown Heights, New York, November 1990

[6] J.-M. Couveignes, *Computing a square root for the number field sieve*, Preprint 1992

[7] D. E. Knuth, *The Art of Computer Programming, vol. 2*, Second Edition, Addison Wesley

[8] Jürgen Loho, *Die Implementierung des Number Field Sieve*, Diplomarbeit an der Universität des Saarlandes 1991

[9] A. K. Lenstra, H. W. Lenstra, M. S. Manasse, J. M. Pollard, *The number field sieve*, Abstract: Proc. 22nd Ann. ACM Symp. on Theory of Computing (STOC)(1990), 564-572

[10] A. K. Lenstra, M. Manasse, *Factoring with two Largeprimes*, Preprint 1992

[11] Thomas Papanikolaou, *libF eine lange Gleitpunktarithmetik*, Diplomarbeit an der Universität des Saarlandes 1993

[12] Pohst & Zassenhaus, *Algorithmic algebraic number theorie*, Encyclopedia of mathematics and its applications, Cambridge University Press

[13] J.M. Pollard, *The Lattice Sieve*, Manuskript, September 1991