Ein Algorithmus zur Bestimmung der Klassengruppe positiv definiter binärer quadratischer Formen

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

Stephan Düllmann

Saarbrücken
1991

Mein besonderer Dank gilt Herrn Professor Buchmann. Er gab die Anregung, an diesem interessanten Thema zu arbeiten. Die vielen fruchtbaren Diskussionen mit ihm waren stets eine motivierende und hilfreiche Unterstützung. Seine Ratschläge haben mich immer weitergebracht.

Ferner danke ich Ralf Roth für die Bereitschaft, die von ihm programmierten Funktionen zur verteilten Anwendung an die Bedürfnisse der Klassengruppenberechnung anzupassen.

Schließlich danke ich meiner Frau und meinen Eltern sowie Claudia und Heiner Dehling und Ingrid Biehl für das Korrekturlesen.

Inhaltsverzeichnis

Kapitel 1

Einführung

1.1 Einleitung

Gegenstand der vorliegenden Arbeit ist die Berechnung imaginär-quadratischer Klassengruppen.

Eine imaginär-quadratische Klassengruppe ist die Gruppe der Äquivalenzklassen binärer quadratischer Formen fester negativer Diskriminante Δ. Diese Gruppe, für die wir kurz $Cl(\Delta)$ schreiben, ist eine endliche abelsche Gruppe, ihre Ordnung wird als die Klassenzahl $h(\Delta)$ bezeichnet. (Auf die Definition der einzelnen Begriffe wird später genauer eingegangen.)

Wir beschreiben einen probabilistischen Algorithmus zur Klassengruppenberechnung, der von uns entwickelt und implementiert wurde und sich in der Praxis als äußerst effizient erwiesen hat. Unser Algorithmus CLASSGROUP berechnet zu gegebener Diskriminante $\Delta < 0$ die Klassenzahl $h(\Delta)$ sowie die Struktur und ein minimales Erzeugendensystem der Klassengruppe $Cl(\Delta)$.

Unter der "Struktur der Klassengruppe" verstehen wir die Darstellung der Klassengruppe als direktes Produkt zyklischer Untergruppen, deren Ordnungen sich sukzessive teilen. Eine solche Darstellung existiert nach dem Hauptsatz für endliche abelsche Gruppen. Unter einem "minimalen Erzeugendensystem" verstehen wir ein System von je einem Erzeuger für jede der zyklischen Untergruppen. Wir werden auch darauf später noch genauer eingehen.

Schon Gauß beschäftigte sich in seinen *Disquisitiones Arithmeticae* [17] eingehend mit der Theorie der binären quadratischen Formen und der quadratischen Klassengruppen. Seit dieser Zeit ist das Problem der Klassengruppenberechnung Gegenstand mathematischer Forschung.

Das Interesse an der Berechnung imaginär-quadratischer Klassenzahlen und Klassengruppen, insbesondere für große Diskriminanten, hat in den letzten Jahren stark zugenommen. Einige Gründe dafür sollen hier kurz erwähnt werden.

In einer Arbeit von Cohen und Lenstra [12] sind eine Reihe heuristischer Resultate über Klassenzahlen und Klassengruppen enthalten. Eines dieser Resultate besagt z.B., daß der ungerade Anteil imaginär-quadratischer Klassengruppen mit fast 98%iger Wahrscheinlichkeit zyklisch ist. Die vorhandenen Tabellen imaginär-quadratischer Klassengruppen [10] bestätigen diese Heuristiken. Umfangreicheres Datenmaterial, insbesondere auch für große Diskriminanten, würde eine weitergehende Bewertung ermöglichen.

Weiterhin gibt es einen Zusammenhang zwischen dem Problem der Klassenzahlberechnung und dem der Faktorisierung großer Zahlen. Sowohl Shanks [39] als auch Seysen [38] haben Algorithmen zur Faktorisierung großer ganzer Zahlen vorgeschlagen, die (in unterschiedlicher Weise) die Arithmetik in imaginär-quadratischen Klassengruppen ausnutzen. Fortschritte im Bereich der Klassengruppenberechnung könnten auch zu einer Verbesserung der Laufzeit dieser Faktorisierungsalgorithmen führen.

Der dritte und wohl wichtigste Grund für das aktuelle Interesse an der Klassengruppenberechnung ist die Beziehung zur Kryptographie. In [8] wird von Buchmann und Williams ein Verfahren zum öffentlichen Schlüsselaustausch vorgestellt, das auf der Arithmetik in imaginär-quadratischen Klassengruppen aufbaut. Die Effizienz des Verfahrens wird in [7] und [16] durch Implementierung und praktische Erprobung nachgewiesen. Die Sicherheit des Verfahrens beruht darauf, daß die Berechnung diskreter Logarithmen in der Klassengruppe nicht effizient durchführbar ist, wenn man die Diskriminante genügend groß wählt. So wird in [8] das Verfahren bei der Verwendung von Diskriminanten mit mindestens 200 Dezimalstellen, die noch einigen weiteren technischen Bedingungen genügen, als sicher angesehen. Da die Berechnung diskreter Logarithmen in der Klassengruppe eng mit der Berechnung der Klassenzahl zusammenhängt, würden genauere Erkenntnisse über die praktische Effizienz von Algorithmen zur Klassenzahlberechnung für große Diskriminanten auch eine bessere Bewertung der Sicherheit dieses Verfahrens ermöglichen.

Der für lange Zeit beste bekannte Algorithmus zur Berechnung imaginär-quadratischer Klassenzahlen stammt von Shanks [39], [40] aus dem Jahre 1972. Es handelt sich um einen deterministischen Algorithmus, der aufgrund seiner Vorgehensweise in der Literatur auch als "baby step - giant step" Algorithmus bezeichnet wird. Der Algorithmus hat für vorgegebene Diskriminante $\Delta < 0$ die Laufzeit $O(|\Delta|^{1/4+\epsilon})$ für jedes $\epsilon > 0$, ist also exponentiell in der Länge der Eingabe. Unter Annahme der Verallgemeinerten Riemannschen Vermutung kann darüber hinaus eine Laufzeit von $O(|\Delta|^{1/5+\epsilon})$ für jedes $\epsilon > 0$ nachgewiesen werden. Die Analyse der Laufzeit findet man in [28] und [37].

In [18] beschreiben und analysieren Hafner und McCurley einen Algorithmus zur Berechnung imaginär-quadratischer Klassengruppen, der nur noch subexponentielle Laufzeit hat. Es handelt sich hierbei um einen probabilistischen Algorithmus vom Typ Las-Vegas. Algorithmen dieses Typs finden nicht mit Sicherheit in jedem Fall ein Ergebnis, wenn sie jedoch ein Resultat ausgeben, so ist dieses mit Sicherheit korrekt. Der Algorithmus berechnet die Klassenzahl und die Struktur der Klassengruppe zu vorgegebener Diskriminante $\Delta < 0$ in erwarteter Laufzeit $L(|\Delta|)^{\sqrt{2}+o(1)}$, wobei $L(|\Delta|) = \exp(\sqrt{\log|\Delta| \log\log|\Delta|})$.

Der von uns entwickelte neue Algorithmus CLASSGROUP benutzt ebenfalls die Grundidee von Hafner und McCurley. Die Motivation zur Entwicklung eines neuen

Algorithmus ist dadurch gegeben, daß der Algorithmus von Hafner und McCurley in der Praxis eine Reihe von Effizienzproblemen verursacht. Unser Algorithmus umgeht diese Probleme und hat sich in der Praxis als sehr effizient erwiesen.

1.2 Gliederung der Arbeit

Im weiteren Verlauf dieses Kapitels werden zunächst einige Bezeichnungen vereinbart, die in den folgenden Kapiteln nicht explizit eingeführt werden. Anschließend wird die in dieser Arbeit verwendete Darstellung der Algorithmen erläutert.

Kapitel 2 beschäftigt sich ausführlich mit den Grundlagen imaginär-quadratischer Klassengruppen. Zuerst werden die in diesem Zusammenhang wichtigen Begriffe eingeführt. Danach werden einige Algorithmen angegeben, die für die effiziente Realisierung der Arithmetik in der Klassengruppe benötigt werden.

In Kapitel 3 werden Algorithmen zur Berechnung gewisser Gitter- bzw. Matrixnormalformen dargestellt. Diese Algorithmen stehen zwar nicht in unmittelbarem Zusammenhang zu Klassengruppen, spielen aber bei der Klassengruppenberechnung eine große Rolle.

In Kapitel 4 wird die grundsätzliche Idee zur Klassengruppenberechnung erläutert, auf der sowohl der Algorithmus von Hafner und McCurley als auch der von uns entwickelte Algorithmus CLASSGROUP aufbauen. Zudem werden einige spezielle Resultate aus der Literatur aufgeführt, die im Zusammenhang mit der Realisierung dieser Idee von Bedeutung sind. Ferner werden zwei Algorithmen aus der elementaren Zahlentheorie angegeben, die später benutzt werden.

In Kapitel 5 wird der Algorithmus von Hafner und McCurley vorgestellt. Dabei werden auch die Resultate aus [18] über Erfolgswahrscheinlichkeit und Laufzeit des Algorithmus nachvollzogen.

In Kapitel 6 beginnt die Beschreibung des Algorithmus CLASSGROUP. Wir geben hier die Effizienzprobleme an, die bei Verwendung des Algorithmus von Hafner und McCurley in der Praxis auftreten, und motivieren so die Entwicklung einer neuen Strategie. Ferner stellen wir die grundsätzlichen neuen Ideen dar, die in die Entwicklung des Algorithmus CLASSGROUP eingeflossen sind, und formulieren den Algorithmus.

In den Kapiteln 7 bis 11 werden die einzelnen Schritte des Algorithmus CLASSGROUP ausführlich beschrieben. Die Beschreibung wird durch Tabellen ergänzt, in denen die Abhängigkeit wichtiger Parameter des Algorithmus von der Größe der Diskriminante verdeutlicht wird.

In Kapitel 12 wird eine Variante des Algorithmus CLASSGROUP erläutert, bei der der zeitaufwendigste Teil des Algorithmus so parallelisiert wird, daß mehrere Rechner gleichzeitig zur Berechnung beitragen können. Mit dieser Variante des Algorithmus wurden Klassenzahl und Klassengruppe für eine Diskriminante mit 55 Dezimalstellen berechnet.

Im Anhang wird eine umfangreiche Klassengruppentabelle angegeben, die mit dem Algorithmus CLASSGROUP berechnet wurde. Sie enthält Klassenzahlen und Klassengruppen für eintausend Diskriminanten mit jeweils 30 Dezimalstellen.

1.3 Bezeichnungen

Zur Bezeichnung der Zahlbereiche verwenden wir die hier angegebenen Symbole:

$\mathbb{N}$	Menge der natürlichen Zahlen (ohne Null)
$\mathbb{N}_0$	Menge der natürlichen Zahlen einschließlich Null
$\mathbb{P}$	Menge der Primzahlen
$\mathbb{Z}$	Menge der ganzen Zahlen
$\mathbb{Q}$	Menge der rationalen Zahlen
$\mathbb{R}$	Menge der reellen Zahlen

Weiterhin vereinbaren wir die folgenden Bezeichnungen, die in den späteren Kapiteln nicht explizit eingeführt werden:

Für $a, b \in \mathbb{Z}$ mit $a \leq b$ bezeichnet $[\![a, b]\!]$ die Menge $\{a, \ldots, b\} \subset \mathbb{Z}$.

Für eine endliche Menge M bezeichnet $\#M$ die Anzahl der Elemente in M. Ist G eine endliche Gruppe, so ist also $\#G$ die Ordnung von G.

Für eine Gruppe G und ein Element $g \in G$ bezeichnet $\langle g \rangle$ die von g erzeugte Untergruppe in G.

Für $a \in \mathbb{Q}, \mathbb{R}$ bezeichnet $\lfloor a \rfloor$ die größte ganze Zahl z mit $z \leq a$.

Für eine Matrix $A \in \mathbb{Z}^{n \times m}$ bezeichnet - wie üblich - a_{ij} $(1 \leq i \leq n,\ 1 \leq j \leq m)$ den Matrixeintrag in der i-ten Zeile und j-ten Spalte. Entsprechend bezeichnet v_i $(1 \leq i \leq n)$ den i-ten Eintrag eines Vektors $v \in \mathbb{Z}^n$.

Zusätzlich führen wir die Bezeichnung A_j $(1 \leq j \leq m)$ für die j-te Spalte einer Matrix $A \in \mathbb{Z}^{n \times m}$ ein.

1.4 Darstellung der Algorithmen

Zur Darstellung der Algorithmen verwenden wir eine modifizierte Form der von Nassi und Shneiderman [31] eingeführten Struktogramme (Nassi-Shneiderman-Diagramme). In diesen Diagrammen wird der Ablauf der Algorithmen durch die Aneinanderreihung und Verschachtelung einzelner sogenannter Strukturblöcke dargestellt.

Der einfachste Strukturblock ist der Anweisungsblock, in dem eine einzelne Anweisung bzw. Operation des Algorithmus enthalten ist:

Anweisung	(1)

Teilweise werden mehrere eng zusammengehörende Anweisungen bzw. Operationen in einem Anweisungsblock zusammengefaßt.

Zur Darstellung von Zählschleifen steht das folgende Konstrukt zur Verfügung:

FOR $v = a$ TO e		(1)
	Anweisung	(2)

Hierbei ist v die Zählvariable, a und e sind Konstanten bzw. Variablen mit $a \leq e$. Dieses Schleifenkonstrukt wird nur verwendet, wenn der Wert der Zählvariablen v sowie, falls es sich nicht um Konstanten handelt, auch der Wert von a bzw. e innerhalb des eigentlichen Schleifenkörpers nicht verändert wird. Schleifen, in denen abwärts gezählt wird, werden entsprechend konstruiert, indem das Wort TO durch DOWNTO ersetzt wird.

Zählschleifen, deren Schleifenkörper sehr einfach ist, werden komplett in einem Anweisungsblock zusammengefaßt:

FOR $v = a$ TO e DO Anweisung.	(1)

Schleifen, deren Fortsetzung bzw. Beendigung von einer Bedingung abhängt, werden mit den folgenden Strukturblöcken dargestellt:

WHILE Bedingung		(1)
	Anweisung	(2)

	Anweisung	(1)
UNTIL Bedingung		(2)

	Anweisung 1	(1)
BREAKIF Bedingung		(2)
	Anweisung 2	(3)

Die WHILE-Schleife und die UNTIL-Schleife sind üblich und bedürfen keiner weiteren Erläuterung. Die BREAKIF-Schleife (Schleife mit Mittelausstieg) wurde aus Gründen der Vereinfachung eingeführt. Sie ist äquivalent zur folgenden Konstruktion:

Anweisung 1		(1)
WHILE NOT Bedingung		(2)
	Anweisung 2	(3)
	Anweisung 1	(4)

Für die Darstellung von Verzweigungen wird das folgende Konstrukt verwendet:

IF Bedingung		(1)
THEN	Anweisung 1	(2)
ELSE	Anweisung 2	(3)

Ist der ELSE-Zweig leer, so wird er weggelassen.

Die hier verwendete Darstellung der Verzweigung unterscheidet sich grundlegend von der üblicherweise in Struktogrammen verwendeten zweispaltigen Form. Die zweispaltige Form ist jedoch nur unter technischen Schwierigkeiten in TEX zu übertragen und benötigt zudem sehr viel Platz.

Falls der ELSE-Zweig leer ist und zusätzlich die Anweisung des THEN-Zweiges sehr einfach ist, kann die gesamte Verzweigung in einem Anweisungsblock zusammengefaßt werden:

IF Bedingung THEN Anweisung.	(1)

Neben den bisher erläuterten Strukturblöcken verwenden wir noch die Anweisung EXIT, mit der der dargestellte Algorithmus bzw. Teilalgorithmus vorzeitig verlassen wird.

Kapitel 2

Imaginär-quadratische Klassengruppen

Dieses Kapitel befaßt sich detailliert mit den Grundlagen imaginär-quadratischer Klassengruppen.

In den ersten beiden Abschnitten werden alle wichtigen Grundbegriffe zusammengestellt. Die Arithmetik in der Klassengruppe wird erläutert, und ihre Komplexität wird angegeben. Für weitere Einzelheiten verweisen wir auf [11], wo (mit Ausnahme der Resultate zur Komplexität) auch die Beweise zu den hier aufgeführten Sätzen zu finden sind. Eine kurze Zusammenstellung der meisten wesentlichen Resultate dieser beiden Abschnitte ist auch in [28] und [37] enthalten.

In den beiden folgenden Abschnitten werden Algorithmen angegeben, die eine effiziente Realisierung der Arithmetik in imaginär-quadratischen Klassengruppen ermöglichen. Dabei werden zunächst im dritten Abschnitt effiziente Versionen der seit langem bekannten Standardalgorithmen beschrieben. Im vierten Abschnitt wird dann ein relativ neuer Algorithmus von Shanks [41] erläutert, mit dem gegenüber den verbesserten Standardalgorithmen aus dem dritten Abschnitt eine weitere Beschleunigung der Arithmetik erreicht wird.

2.1 Quadratische Klassengruppen

Definition 2.1 *Ein Polynom* $f = aX^2 + bXY + cY^2 \in \mathbb{Z}[X,Y]$ *heißt* **binäre quadratische Form***. Die ganze Zahl* $\Delta = b^2 - 4ac$ *heißt* **Diskriminante** *von* f*.* f *heißt* **primitiv***, falls* $\gcd(a,b,c) = 1$*.* f *heißt* **positiv definit***, falls* $\Delta < 0$ *und* $a > 0$*,* **negativ definit***, falls* $\Delta < 0$ *und* $a < 0$*, und* **indefinit***, falls* $\Delta > 0$*.*

Offensichtlich existieren binäre quadratische Formen der Diskriminante Δ genau dann, wenn $\Delta \equiv 0, 1 \bmod 4$. Weiterhin gilt immer $b \equiv \Delta \bmod 2$. Ist die Diskriminante Δ negativ, so müssen die Vorzeichen von a und c übereinstimmen.

Wir betrachten im folgenden nur positiv definite und indefinite binäre quadratische Formen, die außerdem primitiv sind, und bezeichnen diese der Einfachheit halber kurz als "Formen". Weiterhin schließen wir solche Formen aus, deren Diskriminante ein vollständiges Quadrat ist, d.h. wir betrachten nur Formen mit Diskriminanten aus

$$\mathcal{D} = \{\Delta \in \mathbb{Z} \ : \ \Delta \equiv 0, 1 \bmod 4, \ \Delta \text{ kein Quadrat}\}.$$

Für $\Delta \in \mathcal{D}$ bezeichnen wir mit $\mathcal{QF}(\Delta)$ die Menge der Formen mit Diskriminante Δ.

Ferner bezeichnen wir die Form $f = aX^2 + bXY + cY^2$ kurz durch $f = (a, b, c)$. Falls die Diskriminante Δ fixiert ist, schreiben wir noch kürzer $f = (a, b)$, denn c ist durch a, b und Δ eindeutig bestimmt.

Wir führen nun einen Äquivalenzbegriff für Formen ein:

Definition 2.2 *Es seien $f = (a, b, c)$ und $g = (a', b', c')$ Formen. Wir nennen f und g* **äquivalent** *und schreiben dafür $f \sim g$, wenn es eine Matrix*

$$A = \begin{pmatrix} \alpha & \gamma \\ \beta & \delta \end{pmatrix} \in \mathbb{Z}^{2\times 2}$$

mit $\det A = 1$ gibt, so daß

$$a'U^2 + b'UV + c'V^2 = aX^2 + bXY + cY^2$$

ist, wenn man $U = \alpha X + \gamma Y$ und $V = \beta X + \delta Y$ setzt.

Da die Menge der zweireihigen Matrizen mit Determinante 1 eine Gruppe bildet, wird hier tatsächlich eine Äquivalenzrelation definiert. Für eine Form f bezeichnen wir mit $[f]$ die Äquivalenzklasse, in der f liegt.

Man kann leicht verifizieren, daß äquivalente Formen dieselbe Diskriminante haben. Insofern definiert obiger Äquivalenzbegriff nicht nur eine Äquivalenzrelation auf der Menge aller Formen, sondern auch eine Äquivalenzrelation auf der Menge der Formen fester Diskriminante.

Im folgenden sei eine feste Diskriminante $\Delta \in \mathcal{D}$ vorgegeben. Die Menge der Äquivalenzklassen der Formen mit Diskriminante Δ bezeichnen wir mit $Cl(\Delta)$. Die Anzahl dieser Äquivalenzklassen heißt **Klassenzahl** (zur Diskriminante Δ) und wird mit $h(\Delta)$ bezeichnet. Es gilt:

Satz 2.3 *Die Klassenzahl $h(\Delta)$ ist endlich.*

Wir wollen nun eine Gruppenstruktur auf der Menge $Cl(\Delta)$ einführen. Dazu definieren wir zunächst eine Verknüpfung zwischen den Formen der Diskriminante Δ:

Definition 2.4 *Es seien $f_1 = (a_1, b_1), \ f_2 = (a_2, b_2) \in \mathcal{QF}(\Delta)$. Setze*

$$a_3 = \frac{a_1 a_2}{d^2}, \tag{2.1}$$

$$b_3 = v_2 b_1 \frac{a_2}{d} + v_1 b_2 \frac{a_1}{d} + w \frac{b_1 b_2 + \Delta}{2d}, \tag{2.2}$$

wobei

$$d = \gcd\left(a_1, a_2, \frac{b_1+b_2}{2}\right) \tag{2.3}$$

und v_1, v_2, w *die Koeffizienten aus der Darstellung*

$$d = v_1 a_1 + v_2 a_2 + w\frac{b_1+b_2}{2} \tag{2.4}$$

sind. Wir definieren die Form $f_3 = (a_3, b_3) \in \mathcal{QF}(\Delta)$ *als das* **Produkt** *der Formen* f_1 *und* f_2 *und schreiben* $f_3 = f_1 f_2$.

Dieser Produktbegriff ist, wie man leicht nachrechnen kann, mit der oben definierten Äquivalenzrelation verträglich, d.h. für $f_1, f_2, f_3, f_4 \in \mathcal{QF}(\Delta)$ mit $f_1 \sim f_2,\ f_3 \sim f_4$ ist $f_1 f_3 \sim f_2 f_4$. Somit können wir einen Produktbegriff auf $Cl(\Delta)$ definieren:

Definition 2.5 *Es seien* $\mathcal{A}_1, \mathcal{A}_2 \in Cl(\Delta)$ *und* $f_1, f_2 \in \mathcal{QF}(\Delta)$ *mit* $f_1 \in \mathcal{A}_1,\ f_2 \in \mathcal{A}_2$. *Wir definieren das* **Produkt** *von* $\mathcal{A}_1$ *und* $\mathcal{A}_2$ *als*

$$\mathcal{A}_3 = \mathcal{A}_1 \mathcal{A}_2 = [f_1 f_2].$$

Man kann nun folgende Eigenschaften für diese Verknüpfung nachweisen:

- Die Produktbildung auf $Cl(\Delta)$ ist kommutativ und assoziativ.
- Es existiert ein neutrales Element. Für $\Delta \equiv 1 \bmod 4$ ist dies die Klasse $[(1,1)]$, für $\Delta \equiv 0 \bmod 4$ ist es die Klasse $[(1,0)]$. Wir bezeichnen diese Klasse mit 1_{Cl}.
- Zu einer Klasse $[(a,b)]$ ist die Klasse $[(a,-b)]$ das inverse Element.

Somit gilt:

Satz 2.6 *Die Menge* $Cl(\Delta)$ *ist endliche abelsche Gruppe der Ordnung* $h(\Delta)$.

Wir bezeichnen $Cl(\Delta)$ als die **Klassengruppe** (zur Diskriminante Δ).

Nach dem Hauptsatz für endliche abelsche Gruppen (vgl. z.B. [44]) existieren eindeutig bestimmte (nicht notwendigerweise verschiedene) ganze Zahlen $m_1, \ldots, m_t > 1$ mit

$$m_i \mid m_{i+1} \qquad (1 \le i \le t-1), \tag{2.5}$$

so daß

$$Cl(\Delta) \cong \mathbb{Z}/m_1\mathbb{Z} \bigoplus \ldots \bigoplus \mathbb{Z}/m_t\mathbb{Z}. \tag{2.6}$$

$Cl(\Delta)$ ist also direktes Produkt zyklischer Gruppen, deren Ordnungen eindeutig bestimmt sind, wenn man die Teilbarkeitsbedingung (2.5) fordert. Wir bezeichnen das Tupel $(m_1, \ldots, m_t)$ als **die Struktur** der Klassengruppe. Ist $(m_1, \ldots, m_t)$ die Struktur von $Cl(\Delta)$, so gilt offensichtlich

$$h(\Delta) = \prod_{i=1}^{t} m_i. \tag{2.7}$$

Der hier für die Klassengruppe eingeführte Begriff der Struktur läßt sich in derselben Weise auf beliebige endliche abelsche Gruppen verallgemeinern.

Man kann zeigen, daß die Klassengruppe $Cl(\Delta)$ isomorph ist zur Idealklassengruppe der Ordnung der Diskriminante Δ in einem quadratischen Zahlkörper. Auf diesen Zusammenhang soll hier jedoch nicht weiter eingegangen werden. Für Einzelheiten verweisen wir auf [11], [28] und [37]. Aufgrund dieser Zusammenhänge mit quadratischen Zahlkörpern heißen die Klassengruppen zu negativen Diskriminanten (positiv definite Formen) auch **imaginär-quadratische Klassengruppen**, zu positiven Diskriminanten (indefinite Formen) **reell-quadratische Klassengruppen**.

2.2 Reduzierte Formen

Wir beschränken uns nun auf imaginär-quadratische Klassengruppen. Der Grund für diese Spezialisierung ist die Tatsache, daß man in imaginär-quadratischen Klassengruppen auf einfache Weise in jeder Äquivalenzklasse einen eindeutig bestimmten Vertreter auszeichnen kann. Dies erleichtert die Arithmetik in der Klassengruppe. Aus technischen Gründen schließen wir auch $\Delta = -3, -4$ aus. Für die folgenden Betrachtungen sei also die Diskriminante Δ immer aus der Menge

$$\mathcal{D}^- = \{\Delta \in \mathbb{Z}_{<-4} \ : \ \Delta \equiv 0, 1 \bmod 4\},$$

alle Formen seien primitiv und positiv definit.

Wir definieren zunächst den Begriff der reduzierten Form:

Definition 2.7 *Eine Form $f \in \mathcal{QF}(\Delta)$ heißt* **reduziert**, *wenn die beiden folgenden Bedingungen erfüllt sind:*

(i) $|b| \leq a \leq c$,

(ii) $b > 0$, *falls* $|b| = a$ *oder* $a = c$.

Für die Größe des Koeffizienten a einer reduzierten Form gilt

$$a < \sqrt{|\Delta|/3}. \tag{2.8}$$

Aus dieser oberen Schranke ergeben sich sofort auch obere Schranken für die Koeffizienten $|b|$ und c reduzierter Formen. Mit Hilfe dieser oberen Schranken kann später die Komplexität der Produktbildung in der Klassengruppe abgeschätzt werden.

Weiterhin gilt der folgende fundamentale Satz:

Satz 2.8 *Jede Äquivalenzklasse primitiver positiv definiter binärer quadratischer Formen enthält genau eine reduzierte Form.*

Die Berechnung der reduzierten Form in der Äquivalenzklasse einer vorgegebenen Form $f \in \mathcal{QF}(\Delta)$ ist mit dem folgenden Algorithmus leicht möglich.

Algorithmus 2.9 (Reduktion positiv definiter Formen)

Eingabe: positiv definite Form $g = (a', b', c') \in \mathcal{QF}(\Delta)$
Ausgabe: reduzierte Form $f = (a, b, c)$ in der Äquivalenzklasse von g

Setze $f \leftarrow g$, d.h. setze $a \leftarrow a'$, $b \leftarrow b'$, $c \leftarrow c'$.	(1)
Reduziere b modulo $2a$, so daß $\lvert b\rvert < a$ und passe c entsprechend an.	(2)
BREAKIF $\lvert b\rvert \leq a \leq c$.	(3)
Ersetze $f = (a, b, c)$ durch $f' = (c, -b, a)$.	(4)
IF $b < 0$ und entweder $-b = a$ oder $a = c$ THEN setze $b \leftarrow -b$.	(5)

Im Verlauf des Algorithmus wird f in jedem Schritt durch eine äquivalente Form ersetzt. Die Umformungen in Schritt (2) entsprechen der Anwendung der Transformation

$$T_k = \begin{pmatrix} 1 & k \\ 0 & 1 \end{pmatrix}$$

für ein geeignetes $k \in \mathbb{Z}$. Die Veränderungen in Schritt (4) korrespondieren zu der Transformation

$$S = \begin{pmatrix} 0 & 1 \\ -1 & 0 \end{pmatrix}.$$

Die Umformungen in Schritt (5) schließlich entsprechen den Transformationen T_1 bzw. S.

Die Zahl der Schleifendurchläufe im Reduktions-Algorithmus ist nach folgendem Satz beschränkt (Beweis in [26]).

Satz 2.10 *Es sei $(a, b) \in \mathcal{QF}(\Delta)$ eine positiv definite Form. Algorithmus 2.9 benötigt höchstens*

$$O\left(\max\left\{1, \log\left(\frac{a}{\sqrt{|\Delta|}}\right)\right\}\right)$$

Iterationen der Schleife (2) - (4), um die zu (a, b) äquivalente reduzierte Form zu bestimmen.

Mit Hilfe der reduzierten Formen kann man in der Klassengruppe eine effiziente Arithmetik realisieren. Dabei wird jede Klasse durch ihre reduzierte Form repräsentiert. Um das Produkt $\mathcal{A}_3$ zweier Klassen $\mathcal{A}_1$ und $\mathcal{A}_2$ zu berechnen, wird zunächst mit Hilfe der Formeln aus Definition 2.4 das Produkt f_3 der beiden reduzierten Formen aus $\mathcal{A}_1$ und $\mathcal{A}_2$ berechnet. Anschließend wird die Produktform f_3 mit Algorithmus 2.9 reduziert, um die korrekte Repräsentation der Klasse $\mathcal{A}_3 = \mathcal{A}_1\mathcal{A}_2$ zu erhalten.

Da die Koeffizienten reduzierter Formen beschränkt sind, kann man auch für die Koeffizienten der Produktform reduzierter Formen obere Schranken berechnen. Mit Hilfe der Aussage aus Satz 2.10 erhält man dann die folgende Komplexitätsaussage für die Produktberechnung in der Klassengruppe (Beweis in [26]).

Satz 2.11 *Es seien $\mathcal{A}_1, \mathcal{A}_2 \in Cl(\Delta)$ durch ihre reduzierten Formen gegeben. Es gibt einen Algorithmus, der die reduzierte Form in der Klasse $\mathcal{A}_1 \cdot \mathcal{A}_2$ in höchstens*

$$O(\log^2 |\Delta| \log\log |\Delta| \log\log\log |\Delta|)$$

Bit-Operationen berechnet.

2.3 Algorithmen für die Arithmetik in der Klassengruppe

In diesem Abschnitt beschreiben wir Algorithmen, mit denen die Arithmetik in der Klassengruppe effizient realisiert werden kann. Diese Algorithmen wurden in der angegebenen Form bereits in [7] und [16] verwendet. Es sei wiederum eine Diskriminante $\Delta \in \mathcal{D}^-$ fest vorgegeben.

Wir erläutern zunächst den Algorithmus zur Multiplikation von Formen:

Algorithmus 2.12 (Multiplikation von Formen)

Eingabe: zwei Formen $(a_1, b_1), (a_2, b_2) \in \mathcal{QF}(\Delta)$
Ausgabe: das Produkt (a_3, b_3) der beiden Eingabeformen

Berechne mit dem Euklidischen Algorithmus $d' = \gcd(a_1, a_2)$ und den Koeffizienten v' aus der Darstellung $d' = v'a_1 + ua_2$.		(1)
Setze $b_3 \leftarrow v'a_1(b_2 - b_1)$. Setze $a_3 \leftarrow a_1 a_2$.		(2)
IF $d' \neq 1$		(3)
THEN	Berechne mit dem Euklidischen Algorithmus $d = \gcd(d', \frac{b_2+b_1}{2})$ und die Koeffizienten v, w aus der Darstellung $d = vd' + w\frac{b_1+b_2}{2}$.	(4)
	Setze $b_3 \leftarrow \left(b_3 v + w\left(\frac{\Delta - b_1^2}{2}\right)\right)/d$. Setze $a_3 \leftarrow a_3/d^2$.	(5)
Setze $b_3 \leftarrow b_1 + b_3$.		(6)

Die Korrektheit des Algorithmus läßt sich mit Hilfe der Formeln aus Definition 2.4 leicht verifizieren. Neben der Aufteilung der Berechnung von d auf die Schritte (1) und (4) enthält der Algorithmus eine von Shanks [39] vorgeschlagene Verbesserung bei der Berechnung von b_3. Diese besteht darin, in Gleichung (2.2) den Term v_2a_2 durch den Ausdruck

$$v_2a_2 = d - v_1a_1 - w\frac{b_1+b_2}{2}$$

zu ersetzen, der sich aus Gleichung (2.4) ergibt. Damit erhält man für b_3 den Ausdruck

$$\begin{aligned} b_3 &= b_1\left(d - v_1a_1 - w\frac{b_1+b_2}{2}\right)/d + v_1b_2\frac{a_1}{d} + w\frac{b_1b_2+\Delta}{2d} \\ &= b_1 - v_1b_1\frac{a_1}{d} + w\frac{b_1b_2+\Delta-b_1(b_1+b_2)}{2d} + v_1b_2\frac{a_1}{d} \\ &= b_1 + v_1(b_2-b_1)\frac{a_1}{d} + w\frac{\Delta-b_1^2}{2d}, \end{aligned}$$

der in den Schritten (2), (5) und (6) des obigen Algorithmus sukzessive berechnet wird. Durch diese Umformung wird die Bestimmung des Koeffizienten u aus der Darstellung von d' in Schritt (1) sowie eine Multiplikation bei der Berechnung von b_3 eingespart. Man beachte ferner, daß für $d' = 1$ die Schritte (4) und (5) des Algorithmus nicht durchlaufen werden müssen, da in diesem Fall auch $d = 1$ ist und man $v = 1$ sowie $w = 0$ setzen kann.

Sind die zu multiplizierenden Formen gleich, so kann Algorithmus 2.12 bedeutend vereinfacht werden. In diesem Fall ist $d' = a_1 = a_2$, und man kann in Schritt (1) von Algorithmus 2.12 $v' = 0$ sowie $u = 1$ setzen. Damit ergibt sich der im folgenden dargestellte Algorithmus zur Quadrierung von Formen:

Algorithmus 2.13 (Quadrierung von Formen)

Eingabe: eine Form $(a_1, b_1) \in \mathcal{QF}(\Delta)$
Ausgabe: das Quadrat (a_3, b_3) der Eingabeform

Berechne mit dem Euklidischen Algorithmus $d = \gcd(a_1, b_1)$ und den Koeffizienten w aus der Darstellung $d = va_1 + wb_1$.	(1)
Setze $b_3 \leftarrow b_1 + w\frac{\Delta-b_1^2}{2d}$. Setze $a_3 \leftarrow (a_1/d)^2$.	(2)

Es sei noch angemerkt, daß die Algorithmen zur Multiplikation und Quadrierung von Formen auch für Diskriminanten $\Delta \in \mathcal{D}$ gültig sind.

Wir beschreiben eine modifizierte Form des bereits im vorigen Abschnitt angegebenen Algorithmus 2.9 zur Reduktion von Formen. Dazu ist insbesondere die Realisierung von Schritt (2) dieses Algorithmus zu untersuchen. Wir bezeichnen im folgenden die in diesem Schritt jeweils neu berechneten Werte für b und c mit b_N bzw. c_N. Zunächst werden hier ganze Zahlen q und r bestimmt mit $b = q(2a) + r$ und $0 \le r < 2a$. Ist $r > a$, so wird q um 1 erhöht. Anschließend wird

$$b_N = b - q(2a) \tag{2.9}$$

gesetzt und c angepaßt, d.h.

$$c_N = \frac{b_N^2 - \Delta}{4a} \tag{2.10}$$

gesetzt. Nun ist

$$b_N^2 = (b - q(2a))^2 = b^2 - 4aqb + 4a^2q^2.$$

Ferner ist gemäß Gleichung (2.9)

$$q = \frac{b - b_N}{2a},$$

so daß wir

$$b_N^2 = b^2 - 4aqb + 2aq(b - b_N) = b^2 - 2aq(b + b_N)$$

erhalten. Damit können wir Gleichung (2.10) umformen zu

$$c_N = \frac{b^2 - 2aq(b + b_N) - \Delta}{4a} = c - q\frac{b + b_N}{2}.$$

Auf diese Weise kann bei der Berechnung von c_N eine Division durch a eingespart werden. Wir erhalten so den im folgenden dargestellten verbesserten Algorithmus zur Reduktion von Formen.

Algorithmus 2.14 (Reduktion positiv definiter Formen)

Eingabe: positiv definite Form $g = (a', b', c') \in \mathcal{QF}(\Delta)$
Ausgabe: reduzierte Form $f = (a, b, c)$ in der Äquivalenzklasse von g

Setze $f \leftarrow g$, d.h. setze $a \leftarrow a'$, $b \leftarrow b'$, $c \leftarrow c'$.	(1)
Berechne ganze Zahlen q und r, so daß $b = q(2a) + r$ und $0 \leq r < 2a$ ist. Falls $r > a$ ist, erhöhe q um 1.	(2)
Setze $b_N \leftarrow b - q(2a)$. Setze $c \leftarrow c - q\frac{b+b_N}{2}$.	(3)
BREAKIF $\|b_N\| \leq a \leq c$.	(4)
Ersetze $f = (a, b, c)$ durch $f' = (c, -b_N, a)$, d.h. vertausche a und c und setze $b \leftarrow -b_N$.	(5)
Setze $b \leftarrow b_N$.	(6)
IF $b < 0$ und entweder $-b = a$ oder $a = c$ THEN setze $b \leftarrow -b$.	(7)

Mit den in diesem Abschnitt angegebenen Algorithmen kann nun die Arithmetik in der Klassengruppe realisiert werden. Zur Multiplikation zweier Formenklassen werden nacheinander die Algorithmen 2.12 und 2.14 verwendet, zur Quadrierung einer Formenklasse die Algorithmen 2.13 und 2.14.

Im folgenden geben wir noch einen Algorithmus zur Potenzierung von Formenklassen für beliebige Exponenten $n \in \mathbb{N}$ an. Der Algorithmus verwendet die Technik der binären Exponentiation (vgl. [23], S. 442).

Algorithmus 2.15 (Potenzierung in der Klassengruppe)

Eingabe: eine reduzierte Form $g \in \mathcal{QF}(\Delta)$ sowie ein Exponent $n \in \mathbb{N}$
Ausgabe: reduzierte Form f in der Äquivalenzklasse von $[g^n]$

Setze $N \leftarrow n$ und $h \leftarrow g$. Speichere in f die reduzierte Form aus der Klasse $1_{Cl(\Delta)}$, d.h. setze $f \leftarrow (1, \Delta \bmod 4)$.		(1)
	IF N ist ungerade	(2)
	THEN: Setze mit Hilfe von Algorithmus 2.12 $f \leftarrow f \cdot h$ und reduziere dann mit Algorithmus 2.14 die Form f.	(3)
	Setze $N \leftarrow \lfloor N/2 \rfloor$.	(4)
BREAKIF $N = 0$.		(5)
	Setze mit Hilfe von Algorithmus 2.13 $h \leftarrow h^2$ und reduziere dann mit Algorithmus 2.14 die Form h.	(6)

2.4 Die Algorithmen NUCOMP und NUDUPL

In diesem Abschnitt beschreiben wir zwei neue, erst seit kurzem bekannte Algorithmen für die Arithmetik in der Klassengruppe. Auch hier sei $\Delta \in \mathcal{D}^-$ fest vorgegeben.

Im Jahre 1989 veröffentlichte Shanks [41] den neuen Algorithmus NUCOMP, der bei Eingabe zweier Formen $f, g \in \mathcal{QF}(\Delta)$ die reduzierte Form in der Äquivalenzklasse $[f \cdot g]$ berechnet. Mit diesem Algorithmus wird das Auftreten von Zwischenergebnissen der Größe $O(|\Delta|^{3/2})$, das bei Verwendung der bisher bekannten Algorithmen nach Beendigung der Multiplikation möglich ist, vermieden. Die Idee des Algorithmus besteht darin, Multiplikation und Reduktion miteinander zu verbinden, und nicht, wie in den bisher bekannten Algorithmen üblich, nacheinander auszuführen. Dazu werden bereits vor der eigentlichen Multiplikation die Koeffizienten der Ausgangsformen in geeigneter Weise modifiziert und damit bereits ein Teil der Reduktion vorweggenommen. Alle Rechnungen für die Produktbildung in der Klassengruppe werden so mit Zahlen der Größe $O(|\Delta|^{1/2})$ - bzw. höchstens $O(|\Delta|)$ - durchgeführt.

Zur Formulierung des Algorithmus NUCOMP wird zunächst eine modifizierte Form des erweiterten Euklidischen Algorithmus angegeben:

Algorithmus 2.16 (Partieller Euklidischer Algorithmus)

Eingabe: $a, b \in \mathbb{N}$ und eine Konstante $L \in \mathbb{N}$
Ausgabe: $u, d, v_1, v_3 \in \mathbb{Z}$ mit $au \equiv d \bmod b$, $av_1 \equiv v_3 \bmod b$ und $d, v_3 \approx L$

Setze $u \leftarrow 1$, $d \leftarrow a$, $v_3 \leftarrow b$ und $v_1 \leftarrow 0$.		(1)
WHILE $v_3 > L$		(2)
	Setze $q \leftarrow \lfloor d/v_3 \rfloor$ und $t_3 \leftarrow d \bmod v_3$ (Division mit Rest).	(3)
	Setze $t_1 \leftarrow u - qv_1$, $u \leftarrow v_1$, $d \leftarrow v_3$, $v_1 \leftarrow t_1$ und $v_3 \leftarrow t_3$.	(4)

Die Schleife (2) - (4) terminiert hier also nicht erst, wenn in Schritt (2) $v_3 = 0$ ist, wie es im normalen erweiterten Euklidischen Algorithmus der Fall ist, sondern bereits, wenn $v_3 \leq L$ ist. Diese veränderte Abbruchbedingung in Schritt (2) ist der einzige Unterschied zwischen den beiden Algorithmen.

Die Gültigkeit der beiden oben angegebenen Kongruenzen für die Ausgabedaten u, d, v_1 und v_3 kann man leicht nachrechnen. Sie besteht nach Schritt (1) und bleibt beim Durchlaufen der Schleife (2) - (4) bestehen. Die Forderung, daß bei Beendigung des Algorithmus die Zahlen d und v_3 ungefähr die Größe von L haben sollen, ist nur in einem sehr vagen Sinne zu verstehen, da man im allgemeinen d und v_3 nicht beliebig nahe an L annähern kann.

Wir geben nun den Algorithmus NUCOMP an. Wir setzen dabei eine vorberechnete Konstante $L = \lfloor |\Delta/4|^{1/4} \rfloor$ voraus. Ferner ist zu beachten, daß sich die Koeffizienten der Eingabeformen im Verlauf des Algorithmus verändern können. Wenn dies in der konkreten Anwendung nicht in Kauf genommen werden kann, müssen die Eingabeformen vor Beginn des Algorithmus kopiert werden.

Algorithmus 2.17 (NUCOMP)

Eingabe: zwei Formen $f_1 = (a_1, b_1, c_1)$, $f_2 = (a_2, b_2, c_2) \in \mathcal{QF}(\Delta)$
Ausgabe: reduzierte Form $f_3 = (a_3, b_3, c_3)$ in der Äquivalenzklasse $[f_1 \cdot f_2]$

<table>
<tr><td colspan="3">IF $a_1 < a_2$ THEN vertausche f_1 und f_2.</td><td>(1)</td></tr>
<tr><td colspan="3">Setze $s \leftarrow \frac{1}{2}(b_1 + b_2)$ und $n \leftarrow b_2 - s$.</td><td>(2)</td></tr>
<tr><td colspan="3">Berechne mit dem Euklidischen Algorithmus $d = \gcd(a_1, a_2)$ und die Koeffizienten u, v aus der Darstellung $d = ua_1 + va_2$.</td><td>(3)</td></tr>
<tr><td colspan="3">IF $d \neq 1$ und $d \nmid s$</td><td>(4)</td></tr>
<tr><td rowspan="4">THEN</td><td colspan="2">Berechne mit dem Euklidischen Algorithmus $d_1 = \gcd(s, d)$ und den Koeffizienten u_1 aus der Darstellung $d_1 = u_1 s + u'd$.</td><td>(5)</td></tr>
<tr><td colspan="2">IF $d_1 \neq 1$</td><td>(6)</td></tr>
<tr><td>THEN</td><td>Setze $a_1 \leftarrow a_1/d_1$, $a_2 \leftarrow a_2/d_1$, $s \leftarrow s/d_1$, $d \leftarrow d/d_1$.</td><td>(7)</td></tr>
<tr><td colspan="2">Setze $l \leftarrow -u_1(uc_1 + vc_2) \bmod d$ und $A \leftarrow -u(n/d) + l(a_1/d)$.</td><td>(8)</td></tr>
<tr><td>ELSE</td><td colspan="2">Setze $A \leftarrow -un$ und $d_1 \leftarrow d$.</td><td>(9)</td></tr>
<tr><td colspan="3">Setze $A \leftarrow A \bmod a_1$ und $A_1 \leftarrow a_1 - A$. Falls $A_1 < A$ ist, setze dann noch $A \leftarrow -A_1$.</td><td>(10)</td></tr>
<tr><td colspan="3">Führe Algorithmus 2.16 mit den Eingabedaten A und a_1 und der vorberechneten Konstanten L durch und berechne so u, v_1, d, v_3.</td><td>(11)</td></tr>
<tr><td colspan="3">Setze $b \leftarrow (a_2 d + nu)/a_1$, $Q_1 \leftarrow bv_3$, $Q_2 \leftarrow Q_1 + n$, $f \leftarrow Q_2/d$,
$e \leftarrow (sd + c_2 u)/a_1$, $Q_3 \leftarrow ev_1$, $Q_4 \leftarrow Q_3 - s$ und $g \leftarrow Q_4/u$.</td><td>(12)</td></tr>
<tr><td colspan="3">Setze $a_3 \leftarrow db + ed_1 u$, $c_3 \leftarrow v_3 f + g d_1 v_1$ und $b_3 \leftarrow Q_1 + Q_2 + d_1(Q_3 + Q_4)$.</td><td>(13)</td></tr>
<tr><td colspan="3">Reduziere die Form $f_3 = (a_3, b_3, c_3)$ mit Algorithmus 2.14.</td><td>(14)</td></tr>
</table>

Wir geben an dieser Stelle keine Begründung für die Korrektheit des Algorithmus an. Die dazu notwendige Argumentation ist umfangreich und zudem sehr technisch, so daß sie keine intuitiven Einsichten vermittelt. Wir verweisen stattdessen auf die Darstellung in [41] sowie auf ein Manuskript von Atkin [1].

Man beachte, daß alle während des Algorithmus notwendigen Divisionen über $\mathbb{Z}$ ausführbar sind. Vor der Durchführung von Schritt (14) sind zudem die Koeffizienten der zu reduzierenden Form f_3 bereits relativ klein, so daß in Algorithmus 2.14 nur wenige Iterationen für die endgültige Reduktion benötigt werden. Ferner kann man zeigen, daß alle iterativen Schritte des Algorithmus, d.h. die Schritte (3), (5), (11) und (14), in denen iterative Algorithmen benutzt werden, auf Zahlen der Größe höchstens $\sqrt{|\Delta|}$ durchgeführt werden.

Sind die beiden Eingabeformen gleich, so kann der Algorithmus NUCOMP deutlich vereinfacht werden. Atkin führt diese Vereinfachung in [1] durch. Es ergibt sich der im folgenden dargestellte Algorithmus NUDUPL für die Quadrierung von Formenklassen. Wiederum wird die Konstante $L = \lfloor |\Delta/4|^{1/4} \rfloor$ als vorberechnet vorausgesetzt. Auch hier können sich die Koeffizienten der Eingabeform im Verlauf des Algorithmus verändern.

Algorithmus 2.18 (NUDUPL)

Eingabe: eine Form $f_1 = (a_1, b_1, c_1) \in \mathcal{QF}(\Delta)$
Ausgabe: reduzierte Form $f_3 = (a_3, b_3, c_3)$ in der Äquivalenzklasse $[f_1^2]$

Berechne mit dem Euklidischen Algorithmus $d_1 = \gcd(b_1, a_1)$ und den Koeffizienten u aus der Darstellung $d_1 = ub_1 + u'a_1$.	(1)
IF $d_1 \neq 1$ THEN setze $a_1 \leftarrow a_1/d_1$ und $b_1 \leftarrow b_1/d_1$.	(2)
Setze $A \leftarrow -uc_1 \bmod a_1$ und $A_1 \leftarrow a_1 - A$. Falls $A_1 < A$ ist, setze dann noch $A \leftarrow -A_1$.	(3)
Führe Algorithmus 2.16 mit den Eingabedaten A und a_1 und der vorberechneten Konstanten L durch und berechne so u, v_1, d, v_3.	(4)
Setze $e \leftarrow (c_1 u + b_1 d)/a_1$ und $g \leftarrow (ev_1 - b_1)/u$.	(5)
Setze $a_3 \leftarrow d^2$ und $c_3 \leftarrow v_3^2$.	(6)
Setze $b_3 \leftarrow d_1(ev_1 + ug) + (d + v_3)^2 - a_3 - c_3$.	(7)
Setze $a_3 \leftarrow a_3 + ed_1 u$ und $c_3 \leftarrow c_3 + gd_1 v_1$.	(8)
Reduziere die Form $f_3 = (a_3, b_3, c_3)$ mit Algorithmus 2.14.	(9)

Wir haben sowohl die im vorigen Abschnitt erläuterten Verfahren als auch die Algorithmen NUCOMP und NUDUPL implementiert. In der Praxis hat sich gezeigt, daß, insbesondere für große Diskriminanten, die Algorithmen NUCOMP und NUDUPL schneller sind als die Algorithmen 2.12, 2.13 und 2.14.

Kapitel 3

Gitter- und Matrix-Normalformen

In diesem Kapitel werden Algorithmen zur Berechnung gewisser Normalformen von Matrizen bzw. Gitterbasen vorgestellt. Die Algorithmen werden - abgesehen von einigen Verallgemeinerungen - in der Standardform beschrieben, die aus der Literatur bekannt ist. Dabei wird auch die Laufzeit der Algorithmen angegeben.

Im ersten Abschnitt werden Grundbegriffe im Zusammenhang mit unimodularen Transformationen und Gittern erläutert. Die Beweise der hier angegebenen Resultate findet man in [15] bzw. [22].

Im zweiten Abschnitt wird die sogenannte Hermite-Normalform (HNF) definiert und ein Algorithmus zu ihrer Berechnung angegeben. Um für große Matrizen die HNF effizient berechnen zu können, muß dieser Algorithmus modifiziert werden. Dies führt zum Konzept der modularen Hermite-Reduktion, das im dritten Abschnitt beschrieben wird.

Im vierten Abschnitt wird dann der Begriff der Smith-Normalform (SNF) definiert und ein Algorithmus zu ihrer Berechnung angegeben.

Im fünften Abschnitt schließlich wird beschrieben, wie man mit Hilfe von modularer Gauß-Elimination und Chinesischem Restsatz die Determinante großer ganzzahliger Matrizen berechnen kann.

3.1 Theoretische Grundlagen

Eine Matrix $U \in \mathbb{Z}^{n\times n}$ wird als **unimodular** bezeichnet, wenn sie über $\mathbb{Z}$ invertierbar ist. Offensichtlich ist U genau dann unimodular, wenn $|\det U| = 1$ ist. Die Menge der unimodularen $(n \times n)$-Matrizen wird mit $\mathcal{SL}(n)$ bezeichnet. Sie bildet (bezüglich der üblichen Matrix-Multiplikation) eine Gruppe.

Die folgenden Operationen auf einer Matrix $A \in \mathbb{Z}^{n\times m}$ heißen **elementare Spaltentransformationen**:

- Vertauschen zweier Spalten von A,
- Multiplizieren einer Spalte von A mit (-1),
- Addieren eines ganzzahligen Vielfachen einer Spalte von A zu einer anderen Spalte von A.

Jede Folge elementarer Spaltentransformationen wird als **unimodulare Spaltentransformation** bezeichnet. Entsprechend definiert man die Begriffe **elementare Zeilentransformation** und **unimodulare Zeilentransformation**.

Die Multiplikation einer Matrix $A \in \mathbb{Z}^{n \times m}$ mit einer unimodularen Matrix von rechts (bzw. links) ist äquivalent zur Durchführung einer Folge unimodularer Spalten- (bzw. Zeilen-) transformationen auf A. Genauer gilt:

Satz 3.1 *Es seien $A, B \in \mathbb{Z}^{n \times m}$. Dann gilt:*

- *A geht genau dann durch unimodulare Spaltentransformation in B über, wenn es eine Matrix $V \in \mathcal{SL}(m)$ gibt mit $B = AV$.*
- *A geht genau dann durch unimodulare Zeilentransformation in B über, wenn es eine Matrix $U \in \mathcal{SL}(n)$ gibt mit $B = UA$.*
- *A geht genau dann durch eine Folge unimodularer Spalten- und Zeilentransformationen in B über, wenn es Matrizen $U \in \mathcal{SL}(n)$ und $V \in \mathcal{SL}(m)$ gibt mit $B = UAV$.*

Wir führen nun den Begriff des Gitters im $\mathbb{Z}^n$ ein und geben einige wichtige Resultate im Zusammenhang mit Gittern an.

Definition 3.2 *Eine Untergruppe Γ der additiven Gruppe $\mathbb{Z}^n$ heißt* **Gitter im $\mathbb{Z}^n$**. *Die maximale Anzahl linear unabhängiger Vektoren in Γ heißt* **Dimension** *des Gitters und wird mit $\dim \Gamma$ bezeichnet.*

Satz 3.3 *Es sei Γ ein Gitter im $\mathbb{Z}^n$. Der Index von Γ in $\mathbb{Z}^n$ ist genau dann endlich, wenn $\dim \Gamma = n$ ist.*

Definition 3.4 *Es sei Γ ein n-dimensionales Gitter im $\mathbb{Z}^n$. Der Index von Γ in $\mathbb{Z}^n$ heißt* **Determinante** *des Gitters Γ und wird mit $Det\,\Gamma$ bezeichnet.*

Der Begriff der Gitterdeterminante kann auch geometrisch interpretiert werden. Die Determinante eines Gitters ist das n-dimensionale Volumen einer Elementarmasche des Gitters.

Im folgenden bezeichnet E_i $(1 \leq i \leq n)$ den i-ten Einheitsvektor im $\mathbb{Z}^n$.

Satz 3.5 *Es sei Γ ein n-dimensionales Gitter im $\mathbb{Z}^n$ und d ein Vielfaches von $Det\,\Gamma$. Dann ist $dE_i \in \Gamma$ $(1 \leq i \leq n)$.*

Definition 3.6 *Sind* Γ_1, Γ_2 *Gitter im* $\mathbb{Z}^n$ *und ist* $\Gamma_2 \subseteq \Gamma_1$, *so heißt* Γ_2 **Teilgitter** *von* Γ_1.

Satz 3.7 *Sind* Γ_1, Γ_2 n*-dimensionale Gitter im* $\mathbb{Z}^n$ *und ist* Γ_2 *Teilgitter von* Γ_1, *so gilt* $\mathit{Det}\,\Gamma_1 \mid \mathit{Det}\,\Gamma_2$.

Definition 3.8 *Es seien* $v_1, \ldots, v_m$ *Vektoren im* $\mathbb{Z}^n$. *Dann bezeichnen wir mit*

$$\Gamma = \langle v_1, \ldots, v_m \rangle = \left\{ \sum_{i=1}^{n} \lambda_i v_i \;:\; \lambda_i \in \mathbb{Z} \right\}$$

das von diesen Vektoren erzeugte Gitter im $\mathbb{Z}^n$.

Für eine Matrix $A \in \mathbb{Z}^{n \times m}$ bezeichnen wir mit $\Lambda(A)$ das von den Spalten der Matrix A erzeugte Gitter, setzen also

$$\Lambda(A) = \langle A_1, \ldots, A_m \rangle.$$

Wir bezeichnen dieses Gitter kurz als das **von A erzeugte Gitter** im $\mathbb{Z}^n$. Die Dimension des von A erzeugten Gitters heißt **Rang** der Matrix A und wird mit $\operatorname{rang} A$ bezeichnet. Offensichtlich ist $\operatorname{rang} A \le \min\{n, m\}$.

Wir bezeichnen ferner eine Matrix $A \in \mathbb{Z}^{n \times m}$ als **regulär**, wenn $\operatorname{rang} A = n$ ist, andernfalls als **singulär**. Eine quadratische Matrix ist demnach genau dann singulär, wenn ihre Determinante den Wert Null hat. Insofern handelt es sich hier lediglich um eine Verallgemeinerung der für quadratische Matrizen üblichen Bezeichnungen auf nicht-quadratische Matrizen.

Ist $A \in \mathbb{Z}^{n \times n}$ regulär, also $\dim \Lambda(A) = n$, so bilden die Spalten von A eine Basis für das Gitter $\Lambda(A)$. Wir sagen in diesem Fall der Einfachheit halber auch, A ist **Basis(matrix)** für $\Lambda(A)$.

Satz 3.9 *Es seien* $A, B \in \mathbb{Z}^{n \times m}$. *Dann ist* $\Lambda(A) = \Lambda(B)$ *genau dann, wenn es eine Matrix* $K \in \mathcal{SL}(m)$ *gibt, so daß* $AK = B$ *ist.*

Unimodulare Spaltentransformationen auf einer Matrix ändern also das von dieser Matrix erzeugte Gitter nicht. Hieraus folgt insbesondere, daß zwei Basismatrizen eines Gitters betragsmäßig dieselbe Determinante haben. Weiterhin gilt:

Satz 3.10 *Ist* $A \in \mathbb{Z}^{n \times n}$ *regulär, so gilt* $|\det A| = \mathit{Det}\,\Lambda(A)$.

Satz 3.11 *Eine Matrix* $U \in \mathbb{Z}^{n \times n}$ *ist unimodular genau dann, wenn sie eine Basis für das Gitter* $\mathbb{Z}^n$ *ist.*

Satz 3.12 *Ist* $A \in \mathbb{Z}^{n \times m}$ *regulär, so gilt für jede reguläre Teilmatrix* $A' \in \mathbb{Z}^{n \times n}$ *von* A:

- A' *erzeugt ein* n*-dimensionales Teilgitter von* $\Lambda(A)$.
- $\mathit{Det}\,\Lambda(A) \;\Big|\; |\det A'|$.

3.2 Die Hermite-Normalform

Definition 3.13 *Eine Matrix $H \in \mathbb{Z}^{n\times n}$ hat* **Hermite-Normalform (HNF)**, *wenn gilt:*

$$\begin{aligned} h_{ij} &= 0 \qquad (1 \le i < j \le n), \\ h_{ii} &> 0 \qquad (1 \le i \le n), \\ 0 \le h_{ij} &< h_{ii} \qquad (1 \le j < i \le n). \end{aligned}$$

Bereits Hermite hat bewiesen (vgl. [15]):

Satz 3.14 *Zu jeder regulären Matrix $A \in \mathbb{Z}^{n\times n}$ gibt es eine Matrix $K \in \mathcal{SL}(n)$, so daß $H = AK$ Hermite-Normalform hat.*

Weiterhin gilt (Beweis in [15]):

Satz 3.15 *Sind $G, H \in \mathbb{Z}^{n\times n}$ in HNF mit $\Lambda(G) = \Lambda(H)$, so ist $G = H$.*

Aus beiden Sätzen gemeinsam folgt, daß es zu jedem n-dimensionalen Gitter im $\mathbb{Z}^n$ eine eindeutig bestimmte Basis in HNF gibt. Wir geben nun einen Algorithmus an, der diese Basis berechnet, wenn ein Erzeugendensystem des Gitters gegeben ist. Es handelt sich um die Verallgemeinerung eines Algorithmus von Bradley [3] auf nicht-quadratische Matrizen.

Aus technischen Gründen werden alle Algorithmen in diesem Kapitel so formuliert, daß Ein- und Ausgabematrix verschiedene Bezeichnungen haben. Dies erfordert das Kopieren der kompletten Eingabematrix zu Beginn der Algorithmen und führt dazu, daß die Eingabematrix im Verlauf der Algorithmen nicht verändert wird. In der Praxis ist es oft nicht erforderlich, daß die Eingabematrix später unverändert zur Verfügung steht. In diesen Fällen kann der erwähnte Kopiervorgang unterbleiben.

Algorithmus 3.16 (Hermite-Reduktion)

Eingabe: $A \in \mathbb{Z}^{n\times m}$ mit $m \ge n$ und rang $A = n$
Ausgabe: $H \in \mathbb{Z}^{n\times n}$ in HNF, so daß H Basis von $\Lambda(A)$ ist

Setze $H \leftarrow A$.	(1)
/* Bringe H auf Dreiecksgestalt mit positiven Diagonaleinträgen. */	
FOR $i = 1$ TO n	(2)
Führe elementare Spaltenoperationen auf den Spalten i bis m von H durch, so daß $h_{ii} > 0$ ist sowie $h_{ij} = 0$ für $i < j \le m$.	(3)
Entferne die Spalten $n+1$ bis m aus H.	(4)
/* Reduziere die Einträge unterhalb der Diagonalen. */	
FOR $i = 2$ TO n	(5)
Führe elementare Spaltenoperationen auf den Spalten 1 bis i von H durch, so daß $0 \le h_{ij} < h_{ii}$ für $1 \le j < i$.	(6)

In Schritt (4) können die Spalten $n+1$ bis m aus der Matrix H entfernt werden, ohne daß sich das von H erzeugte Gitter verändert, da diese Spalten nach der Durchführung von Schritt (3) nur noch Nullen enthalten.

Die in Schritt (3) durchzuführenden Spaltenoperationen werden mit dem folgenden Teilalgorithmus realisiert:

Algorithmus 3.17

Eingabe:	$A' \in \mathbb{Z}^{n \times m}$ mit $m \geq n$ und rang $A' = n$ sowie ein Index $i \in [\![1, n]\!]$
Ausgabe:	$A \in \mathbb{Z}^{n \times m}$ mit $\Lambda(A) = \Lambda(A')$, $a_{ii} > 0$ sowie $a_{ij} = 0$ für $i < j \leq m$

Setze $A \leftarrow A'$.		(1)
IF $a_{ii} = 0$ THEN vertausche die i-te Spalte von A mit einer der Spalten A_{i+1} bis A_m, so daß dann $a_{ii} \neq 0$ ist.		(2)
FOR $j = i + 1$ TO m		(3)
	Berechne mit dem Euklidischen Algorithmus $r = \gcd(a_{ii}, a_{ij})$ und die Koeffizienten p, q aus der Darstellung $r = pa_{ii} + qa_{ij}$.	(4)
	Setze (Spalte) $A'_i \leftarrow pA_i + qA_j$. Setze (Spalte) $A'_j \leftarrow \frac{a_{jj}}{r} A_j - \frac{a_{ij}}{r} A_i$. Setze dann $A_i \leftarrow A'_i$ und $A_j \leftarrow A'_j$.	(5)

Die in jedem Durchlauf von Schritt (5) durchgeführte Transformation der Matrix A entspricht der Multiplikation von A mit der Einbettung der Matrix

$$D = \begin{pmatrix} p & -a_{ij}/r \\ q & a_{jj}/r \end{pmatrix}$$

in die m-reihige Einheitsmatrix an den Positionen i und j. Da D unimodular ist, handelt es sich hierbei um unimodulare Spaltentransformationen. Ein- und Ausgabematrix erzeugen also dasselbe Gitter.

Bei jedem Durchlauf von Schritt (5) wird der Eintrag a_{ij} (rechts von der Diagonalen) ersetzt durch

$$\frac{a_{ii}}{r} a_{ij} - \frac{a_{ij}}{r} a_{ii} = 0,$$

und der Diagonaleintrag a_{ii} wird ersetzt durch

$$pa_{ii} + qa_{ij} = r > 0,$$

die Ausgabematrix hat also die gewünschte Form.

Die in Schritt (6) von Algorithmus 3.16 durchzuführenden Spaltenoperationen werden mit dem folgenden Teilalgorithmus realisiert:

Algorithmus 3.18

Eingabe: $H' \in \mathbb{Z}^{n\times n}$ in unterer Dreiecksform mit positiven Diagonaleinträgen und ein Index $i \in [\![2, n]\!]$

Ausgabe: $H \in \mathbb{Z}^{n\times n}$ in unterer Dreiecksform mit positiven Diagonaleinträgen und $\Lambda(H) = \Lambda(H')$ sowie $0 \le h_{ij} < h_{ii}$ für $1 \le j < i$

Setze $H \leftarrow H'$.		(1)
FOR $j = 1$ TO $i - 1$		(2)
	Setze $q \leftarrow \lfloor h_{ij}/h_{ii} \rfloor$.	(3)
	Setze (Spalte) $H_j \leftarrow H_j - qH_i$.	(4)

Bei den in Schritt (4) durchgeführten Transformationen von H handelt es sich um elementare Spaltentransformationen. Ein- und Ausgabematrix erzeugen also dasselbe Gitter. Bei jedem Durchlauf von Schritt (4) wird der Eintrag h_{ij} (unterhalb der Diagonalen) durch

$$h'_{ij} = h_{ij} - \left\lfloor \frac{h_{ij}}{h_{ii}} \right\rfloor,$$

also seinen kleinsten positiven Rest modulo h_{ii}, ersetzt. Die Ausgabematrix hat somit die gewünschte Form.

Da alle Spaltenoperationen mit $O(n)$ arithmetischen Operationen (Addition, Subtraktion, Multiplikation oder Division ganzer Zahlen) realisierbar sind, erhalten wir die folgende Aussage über die Anzahl der Rechenoperationen für die Hermite-Reduktion.

Satz 3.19 *Es sei $A \in \mathbb{Z}^{n\times m}$, $m \ge n$, rang $A = n$. Algorithmus 3.16 benötigt $O(nm)$ Anwendungen des erweiterten Euklidischen Algorithmus und $O(n^2m)$ arithmetische Operationen, um eine Basis des Gitters $\Lambda(A)$ in HNF zu berechnen.*

Dieser Satz erlaubt natürlich keine Komplexitätsanalyse, da die Größe der auftretenden Zahlen nicht genannt ist. Tatsächlich ist für diesen Algorithmus keine polynomielle Abschätzung für die Größe der Zwischenergebnisse bekannt.

Die Größe der Zwischenergebnisse verursacht auch in der Praxis erhebliche Probleme. Schon bei der Hermite-Reduktion relativ kleiner Matrizen mit kleinen Einträgen können Zwischenergebnisse von enormer Größe auftreten. In [15] wird dazu ein Beispiel angeführt: Bei der Hermite-Reduktion einer (8×8)-Matrix, deren Einträge alle betragsmäßig durch 2^{16} beschränkt sind, treten Zwischenergebnisse über 2^{432} auf. Dieses Problem wird in der Literatur als "intermediate expression swell" oder "entry explosion" bezeichnet. Für größere Matrizen wachsen deshalb Speicherplatz und Rechenzeit so stark an, daß der Algorithmus für die Praxis nicht mehr geeignet ist.

Kannan und Bachem [22] geben einen modifizierten Algorithmus zur Hermite-Reduktion an und zeigen, daß bei diesem Algorithmus sowohl die Zahl der arithmetischen

Operationen als auch die Größe der Zwischenergebnisse polynomiell beschränkt sind. Dieses Resultat ist jedoch eher für die Theorie von Bedeutung, da die polynomielle Schranke sehr groß ist und in der Praxis ebenfalls das Problem der "entry explosion" auftritt.

3.3 Modulare Hermite-Reduktion

Das Problem der "entry explosion" bei der Berechnung von Hermite-Normalformen kann mit Hilfe der sogenannten **modularen Hermite-Reduktion** umgangen werden. Voraussetzung ist die Kenntnis der Determinante des von der Matrix erzeugten Gitters bzw. eines Vielfachen dieser Determinante vor Beginn der Berechnung.

Die Idee der modularen Hermite-Reduktion wurde erstmals von Domich, Kannan und Trotter in [15] vorgestellt. Diese Arbeit beschäftigt sich nur mit quadratischen Matrizen und setzt die Kenntnis der exakten Gitterdeterminante voraus. Hier wird nun der Algorithmus für eine leicht verallgemeinerte Situation dargestellt. Wir behandeln auch nicht-quadratische Matrizen und setzen nur die Kenntnis eines Vielfachen der Gitterdeterminante voraus.

Es sei also $A \in \mathbb{Z}^{n \times m}$, $m \geq n$, rang $A = n$, $\Gamma = \Lambda(A)$ das von A erzeugte Gitter und d ein Vielfaches der Determinante von Γ. Die Grundidee des Algorithmus beruht auf der Tatsache, daß die Vektoren dE_i $(1 \leq i \leq n)$ ebenfalls zu Γ gehören (vgl. Satz 3.5).

Wir führen nun zunächst die Schritte (1) bis (4) von Algorithmus 3.16 durch, wobei wir aber nach jeder in Schritt (3) durchgeführten Spaltenoperation alle neu berechneten Einträge modulo d reduzieren. Hierbei handelt es sich formal um die Addition irgendwelcher Vektoren aus Γ zu den Spalten der Matrix. Der Effekt dieser Veränderung ist, daß die Spalten der entstehenden Matrix zwar sicherlich in Γ liegen, aber nicht mehr notwendigerweise ein Erzeugendensystem für Γ bilden.

Wir konstruieren also zunächst eine Dreiecksmatrix L, deren Spalten in Γ liegen, aber nicht notwendigerweise eine Basis von Γ bilden. Aus L und d wird daher anschließend eine Basis L' von Γ in Dreiecksform rekonstruiert. Dazu wird der im folgenden angegebene Satz benötigt.

Satz 3.20 *Es sei $A \in \mathbb{Z}^{n \times m}$, rang $A = n$. $H = (h_{ij}) \in \mathbb{Z}^{n \times n}$ sei die Basis von $\Lambda(A)$ in HNF, d sei ein positives ganzzahliges Vielfaches von Det $\Lambda(A)$. Setze $d_1 = d$ und $d_{i+1} = d_i / h_{ii}$ $(1 \leq i \leq n-1)$. Dann gilt:*

(i) Ist $L = (l_{ij}) \in \mathbb{Z}^{n \times n}$ untere Dreiecksmatrix, die aus A durch eine Folge unimodularer Spaltenoperationen, gefolgt von Reduktion modulo d, entstanden ist, so ist $h_{ii} = \gcd(d_i, l_{ii})$.

(ii) Es gilt $d_i E_j \in \Lambda(A)$ $(1 \leq i \leq j \leq n)$.

Teil (i) dieses Satzes ermöglicht die Berechnung der Diagonaleinträge von H aus L und d. Er wird in [18] bewiesen. Der Beweis folgt der Argumentation in [15], die für $m = n$ und $d = \text{Det } \Lambda(A)$ gegeben wird.

Teil (ii) wird in [15] ebenfalls für $m = n$ und $d = \text{Det}\,\Lambda(A)$ sowie nur für $i = j$ bewiesen. Die hier angegebene Verallgemeinerung ist eine triviale Folgerung.

Wir berechnen also $t_i \in \mathbb{Z}$ mit $t_i l_{ii} \equiv h_{ii} \bmod d_i$, ersetzen die Spalte L_i von L durch $L_i' = t_i L_i$ und reduzieren alle neu berechneten Einträge modulo d_i. Die Spalten L_i' liegen im Gitter Γ, da die zur Reduktion der Einträge verwendeten Vektoren nach Satz 3.20 (ii) in Γ liegen. Die Dreiecksgestalt der Matrix bleibt dabei offensichtlich erhalten. Weiterhin ist $l_{ii}' = h_{ii}$. Mit dem folgenden Satz, der ebenfalls in [15] bewiesen wird, ergibt sich, daß die so entstandene Matrix eine Basis von Γ ist.

Satz 3.21 *Sei Γ n-dimensionales Gitter im $\mathbb{Z}^n$, $H = (h_{ij}) \in \mathbb{Z}^{n \times n}$ HNF von Γ. Sei $G = (g_{ij}) \in \mathbb{Z}^{n \times n}$ untere Dreiecksmatrix, deren Spalten in Γ liegen und deren Diagonaleinträge $g_{ii} = h_{ii}$ $(1 \leq i \leq n)$ erfüllen. Dann ist G eine Basis für Γ.*

In der nun vorhandenen Basis L' von Γ in Dreiecksform müssen noch die Einträge unterhalb der Diagonalen reduziert werden, um eine Basis in HNF zu erhalten. Dies geschieht wie in der Schleife (5) - (6) von Algorithmus 3.16. Damit auch hier die Zwischenergebnisse nicht zu groß werden, reduziert man dabei bei jeder Spaltenoperation im i-ten Durchlauf der Schleife die Einträge an den Positionen $k = i+1, \dots, n$ der veränderten Spalte modulo d_k. Durch diese Reduktionen werden offenbar die Diagonaleinträge der Matrix nicht verändert. Ferner liegen nach Satz 3.20 (ii) für $1 \leq k < l \leq n$ die Vektoren $d_k E_l$ in Γ. Insofern hat die entstehende Matrix nicht nur HNF, sondern ist nach Satz 3.21 immer noch eine Basis von Γ.

Aus den genannten Überlegungen ergibt sich der folgende Algorithmus:

Algorithmus 3.22 (Modulare Hermite-Reduktion)

Eingabe: $A \in \mathbb{Z}^{n \times m}$ mit $m \geq n$ und rang $A = n$ sowie ein Vielfaches $d \in \mathbb{N}$ von Det $\Lambda(A)$

Ausgabe: $H \in \mathbb{Z}^{n \times n}$ in HNF, so daß H Basis von $\Lambda(A)$ ist

Setze $H \leftarrow A$.	(1)
/* Bringe H auf Dreiecksgestalt mit positiven Diagonaleinträgen. */	
FOR $i = 1$ TO n	(2)
Führe mit Algorithmus 3.17 elementare Spaltenoperationen auf den Spalten i bis m von H durch, so daß $h_{ii} > 0$ ist sowie $h_{ij} = 0$ für $i < j \leq m$. Reduziere dabei bei jeder Spaltenoperation alle neu berechneten Matrixeinträge modulo d.	(3)
Entferne die Spalten $n + 1$ bis m aus H.	(4)
/* Rekonstruiere eine Basis von $\Lambda(A)$. */	
Setze $d_1 \leftarrow d$.	(5)
FOR $i = 1$ TO n	(6)
Berechne mit dem Euklidischen Algorithmus $h_i = \gcd(d_i, h_{ii})$ und den Koeffizienten t aus der Darstellung $h_i = rd_i + th_{ii}$.	(7)
Setze $d_{i+1} \leftarrow d_i / h_i$.	(8)
Setze (Spalte) $H_i \leftarrow tH_i$. Reduziere dabei alle neu berechneten Einträge modulo d_i.	(9)
/* H hat nun Dreiecksgestalt und erzeugt $\Lambda(A)$. Es werden jetzt die Einträge unterhalb der Diagonalen reduziert. */	
FOR $i = 2$ TO n	(10)
Führe mit Algorithmus 3.18 elementare Spaltenoperationen auf den Spalten 1 bis i von H durch, so daß $0 \leq h_{ij} < h_{ii}$ für $1 \leq j < i$. Reduziere dabei bei jeder Spaltenoperation die Einträge an den Positionen $k = i + 1, \ldots, n$ der veränderten Spalte modulo d_k.	(11)

Offensichtlich erhält man für Algorithmus 3.22 dieselbe Aussage über die Anzahl der Rechenoperationen wie für Algorithmus 3.16 (Satz 3.19). Allerdings haben hier alle Zahlen maximal die Größe d.

Mit Hilfe der Techniken zur schnellen Multiplikation kann der erweiterte Euklidische Algorithmus auf Eingaben der Größe d in $O(\log d \log\log^3 d)$ Bit-Operationen durchgeführt werden (siehe [36]). Für alle anderen arithmetischen Operationen auf Zahlen der Größe d werden mit schneller Multiplikation höchstens $O(\log d \log\log^2 d)$ Bit-Operationen benötigt (siehe [35]). Somit erhalten wir die folgende Aussage über die Komplexität der modularen Hermite-Reduktion.

Satz 3.23 *Es gibt einen Algorithmus , der bei Eingabe einer Matrix $A \in \mathbb{Z}^{n \times m}$ mit $m \geq n$ und rang $A = n$ und eines Vielfachen d der Determinante des Gitters $\Lambda(A)$ in*

$$O(nm \log d \log \log^3 d + n^2 m \log d \log \log^2 d)$$

Bit-Operationen die Basis von $\Lambda(A)$ in HNF berechnet.

3.4 Die Smith-Normalform

Definition 3.24 *Eine Matrix $S \in \mathbb{Z}^{n \times n}$ hat* **Smith-Normalform (SNF)**, *wenn gilt:*

$$\begin{aligned} s_{ij} &= 0 \qquad (1 \leq i, j \leq n,\ i \neq j), \\ s_{ii} &> 0 \qquad (1 \leq i \leq n), \\ s_{ii} &\mid s_{i+1,i+1} \qquad (1 \leq i \leq n-1). \end{aligned}$$

Bereits Smith hat bewiesen (vgl. [22]):

Satz 3.25 *Zu jeder regulären Matrix $A \in \mathbb{Z}^{n \times n}$ gibt es Matrizen $U, V \in \mathcal{SL}(n)$, so daß $S = UAV$ Smith-Normalform hat.*

Weiterhin gilt (zum Beweis vgl. [21], S. 344):

Satz 3.26 *Es sei $A \in \mathbb{Z}^{n \times n}$ regulär und es seien $U, V \in \mathcal{SL}(n)$, so daß $S = UAV$ Smith-Normalform hat. Es seien s_i $(1 \leq i \leq n)$ die Diagonaleinträge von S. Dann gilt*

$$\mathbb{Z}^n / \Lambda(A) \ \cong\ \mathbb{Z}/s_1\mathbb{Z} \bigoplus \ldots \bigoplus \mathbb{Z}/s_n\mathbb{Z}.$$

Aus der Smith-Normalform einer regulären Matrix $A \in \mathbb{Z}^{n \times n}$ kann man also die Struktur der endlichen abelschen Gruppe $G = \mathbb{Z}^n/\Lambda(A)$ ablesen. Ist nämlich i der kleinste Index mit $s_i > 1$, so ist $(s_i, \ldots, s_n)$ die Struktur von G. Aus dieser Beobachtung und dem Hauptsatz für endliche abelsche Gruppen folgt insbesondere auch, daß für ein vorgegebenes n-dimensionales Gitter im $\mathbb{Z}^n$ die Smith-Normalform eindeutig bestimmt ist.

Wir geben nun einen Algorithmus zur Berechnung der Smith-Normalform an. Die Grundidee dieses Algorithmus besteht darin, mit Hilfe unimodularer Zeilen- und Spaltentransformationen zunächst die Einträge in der ersten Zeile und Spalte der Matrix (bis auf den Diagonaleintrag a_{11}) zu Null zu machen. Zusätzlich soll a_{11} alle verbleibenden Einträge der Matrix teilen. Der so entstandene Eintrag a_{11} ist dann der erste Diagonaleintrag der zu berechnenden SNF. Anschließend wird dasselbe Verfahren sukzessive für die weiteren Zeilen und Spalten durchgeführt. Bei den dafür erforderlichen Transformationen werden die bereits berechneten Diagonaleinträge nicht mehr verändert.

Ebenso wie bei der Berechnung der HNF werden alle Operationen modular durchgeführt, um die Größe der Zwischenergebnisse zu beschränken. Als Modul wird hier die Determinante der Matrix verwendet. Diese ist insbesondere dann bekannt, wenn die Matrix bereits in HNF vorliegt.

Ähnlich wie bei der modularen Hermite-Reduktion wird zunächst eine Matrix konstruiert, die die gewünschte Form hat (hier Diagonalform). Aus den Diagonaleinträgen und dem Modul werden dann die korrekten Diagonaleinträge der Normalform rekonstruiert. Für die modulare Smith-Reduktion wird dazu der folgende Satz benötigt, der aus der Darstellung in [18] folgt (vgl. auch die Analogie zu Satz 3.20 (i)):

Satz 3.27 *Es seien $A \in \mathbb{Z}^{n\times n}$ regulär, d die Determinante von A, $s_1, \ldots, s_n$ die Diagonaleinträge der SNF von A. Setze $d_1 = d$ und $d_{i+1} = d_i/s_i$ $(1 \le i \le n-1)$. Dann gilt: Sind $t_1, \ldots, t_n$ die Diagonaleinträge einer Diagonalmatrix, die aus A durch eine Folge unimodularer Zeilen- und Spaltenoperationen, gefolgt von Reduktion modulo d, entstanden ist, so ist $s_i = \gcd(d_i, t_i)$.*

Damit ergibt sich der folgende Algorithmus:

Algorithmus 3.28 (Smith-Reduktion)

Eingabe: $A' \in \mathbb{Z}^{n\times n}$ mit rang $A' = n$ sowie $d = \det A'$
Ausgabe: die Diagonaleinträge $s_1, \ldots, s_n \in \mathbb{Z}$ der SNF von A'

- Setze $A \leftarrow A'$. (1)
- /* Bringe A in Diagonalgestalt. */
- FOR $i = 1$ TO n (2)
  - WHILE es gibt einen Eintrag $\neq 0$ in Zeile i rechts von a_{ii} oder in Spalte i unterhalb von a_{ii} (3)
    - Erzeuge mit Algorithmus 3.17 Nullen in Zeile i rechts von der Diagonalen. Reduziere dabei bei jeder Spaltenoperation die neu berechneten Matrixeinträge modulo d. (4)
    - Erzeuge Nullen in Spalte i unterhalb der Diagonalen. Wende dazu Algorithmus 3.17 für den Index i auf die transponierte Matrix A^T an. Reduziere dabei bei jeder Spaltenoperation die neu berechneten Matrixeinträge modulo d. (5)
  - BREAKIF $a_{ii} \mid a_{jk}$ für $i+1 \le j, k \le n$. (6)
    - Addiere Zeile j von A zu Zeile i. (7)
- /* Rekonstruiere die Diagonaleinträge der SNF. */
- Setze $d_1 \leftarrow d$. (8)
- FOR $i = 1$ TO n (9)
  - Berechne mit dem Euklidischen Algorithmus $s_i = \gcd(d_i, a_{ii})$. (10)
  - Setze $d_{i+1} \leftarrow d_i/s_i$. (11)

Alle Veränderungen der Matrix A in Schritt (4) bzw. (5) sind unimodulare Zeilen- bzw. Spaltentranformationen, gefolgt von Reduktion modulo d. Vor Schritt (8) hat A Diagonalgestalt. Somit werden gemäß Satz 3.27 in den Schritten (9) bis (11) die Diagonaleinträge der SNF richtig berechnet. Wir zeigen nun noch, daß die Schleife (3) – (7) und die darin enthaltene innere Schleife (3) – (5) jeweils nach endlich vielen Iterationen terminieren.

Wir betrachten zunächst die innere Schleife (3) – (5). Nach der Durchführung der Spaltenoperationen in Schritt (4) stehen in Zeile i der Matrix rechts von der Diagonalen nur noch Nullen, und der Diagonaleintrag a_{ii} ist der größte gemeinsame Teiler aller vorher in dieser Zeile stehenden Einträge. Anschließend werden in Schritt (5) Zeilenoperationen durchgeführt, mit denen die Einträge der Spalte i unterhalb der Diagonalen zu Null gemacht werden. Der entstehende Diagonaleintrag a_{ii} ist der größte gemeinsame Teiler aller vorher in Spalte i stehenden Einträge. Durch die Zeilenoperationen können die vorher in Zeile i erzeugten Nullen teilweise wieder zerstört werden. Dies geschieht jedoch nur, wenn der Wert von a_{ii} durch die Zeilenoperationen tatsächlich verändert wurde. In diesem Fall ist die Veränderung von a_{ii} eine multiplikative Verkleinerung mindestens um den Faktor 2. Nach höchstens $O(\log d)$ Wiederholungen der Schritte (4) und (5) erhalten wir daher eine Matrix, bei der Zeile i und Spalte i außerhalb der Diagonalen nur Nullen enthalten, und die innere Schleife bricht ab.

Existiert nun noch ein Eintrag a_{jk} $(i < j, k \leq n)$, der von a_{ii} nicht geteilt wird, so wird in Schritt (7) Zeile j zu Zeile i hinzuaddiert, und der Vorgang der Zeilen- und Spaltenoperationen beginnt erneut. Auch in jedem Durchlauf dieser äußeren Schleife (3) – (7) wird a_{ii} um einen multiplikativen Faktor von mindestens 2 verkleinert, so daß nach höchstens $O(\log d)$ Iterationen der Diagonaleintrag a_{ii} jeden Eintrag a_{jk} $(i < j, k \leq n)$ teilt und damit auch diese Schleife beendet ist.

Wir untersuchen nun die Laufzeit von Algorithmus 3.28. Die Schritte (4) und (5) werden höchstens $n \log^2 d$ mal durchlaufen. Jeder Durchlauf erfordert $O(n)$ Anwendungen des erweiterten Euklidischen Algorithmus sowie $O(n)$ Zeilen- bzw. Spaltenoperationen. Schritt (7) wird höchstens $n \log d$ mal durchlaufen und erfordert bei jedem Durchlauf eine Zeilenoperation. In den Schritten (9) bis (11) erfolgen n Anwendungen des Euklidischen Algorithmus und n arithmetische Operationen. Da jede Zeilen- und Spaltenoperation $O(n)$ arithmetische Operationen erfordert, gilt:

Satz 3.29 *Es sei $A \in \mathbb{Z}^{n \times n}$ regulär. Algorithmus 3.28 benötigt $O(n^2 \log^2 d)$ Anwendungen des erweiterten Euklidischen Algorithmus und $O(n^3 \log^2 d)$ arithmetische Operationen, um die SNF von A zu berechnen.*

Da während der Berechnung alle Zahlen maximal die Größe d haben, erhalten wir unter Benutzung derselben Aussagen über die Komplexität der einzelnen Rechenoperationen wie für Satz 3.23 den folgenden Satz.

Satz 3.30 *Es gibt einen Algorithmus, der in*

$$O(n^2 \log^3 d \log\log^3 d + n^3 \log^3 d \log\log^2 d)$$

Bit-Operationen die SNF einer regulären Matrix $A \in \mathbb{Z}^{n \times n}$ berechnet.

3.5 Modulare Gauß-Elimination

Wir erläutern nun die Berechnung der Determinante ganzzahliger quadratischer Matrizen. Der üblicherweise zu diesem Zweck verwendete Algorithmus ist die Gauß-Elimination. Wir geben zunächst einen Algorithmus zur Gauß-Elimination für Matrizen über beliebigen Körpern an.

Algorithmus 3.31 (Gauß-Elimination)

Eingabe: $A' \in K^{n \times n}$ (K Körper)
Ausgabe: $d = \det A' \in K$

Setze $A \leftarrow A'$.	(1)
Setze $d \leftarrow 1$.	(2)
FOR $i = 1$ TO n	(3)
IF $a_{ii} = 0$	(4)
THEN Vertausche die i-te Spalte von A mit einer der Spalten A_{i+1} bis A_n, so daß dann $a_{ii} \neq 0$ ist und setze $d \leftarrow -d$.	(5)
IF keine solche Spalte existiert THEN setze $d \leftarrow 0$ und EXIT.	(6)
Berechne $b \leftarrow (a_{ii})^{-1}$.	(7)
FOR $j = i + 1$ TO n	(8)
Setze (Spalte) $A_j \leftarrow A_j - a_{ij} b A_i$.	(9)
Setze $d \leftarrow d \cdot a_{ii}$.	(10)

Satz 3.32 *Es sei K ein Körper. Algorithmus 3.31 benötigt $O(n)$ Inversenberechnungen und weitere $O(n^3)$ arithmetische Operationen in K, um die Determinante einer Matrix $A \in K^{n \times n}$ zu berechnen.*

Um mit diesem Algorithmus die Determinante einer ganzzahligen Matrix A zu berechnen, muß man A als Matrix über $\mathcal{Q}$ auffassen und alle Rechnungen über $\mathcal{Q}$ ausführen. Die Größe der Zähler und Nenner steigt dabei im Verlauf der Rechnung ständig an; der Algorithmus ist deshalb in dieser Form wegen des hohen Bedarfs an Rechenzeit und Speicherplatz zur Bearbeitung großer ganzzahliger Matrizen nicht geeignet. Man verwendet daher in diesem Fall die sogenannte **modulare Gauß-Elimination**, bei der die Notwendigkeit des Rechnens mit beliebig großen Zahlen umgangen wird.

Zur Erläuterung dieses Verfahrens geben wir zunächst den Chinesischen Restsatz an (Beweis in [23], S. 270).

Satz 3.33 (Chinesischer Restsatz) *Es seien $m_1, \dots m_k \in \mathbb{N}$ paarweise teilerfremd und $x_1, \dots, x_k \in \mathbb{Z}$. Dann existiert genau eine ganze Zahl x mit*

$$0 \leq x < \prod_{i=1}^{k} m_i, \tag{3.1}$$

so daß

$$x \equiv x_i \; mod \; m_i \qquad (1 \leq i \leq k). \tag{3.2}$$

Der Chinesische Restsatz besagt also, daß eine Zahl $x \in [\![0, M]\!]$ (mit $M \in \mathbb{N}$) bereits eindeutig bestimmt ist, wenn man ihre Reste modulo mehrerer teilerfremder Zahlen $m_i \in \mathbb{N}$ kennt, deren Produkt M übersteigt. Zur Ausnutzung dieses Satzes in der Praxis benötigt man einen Algorithmus, mit dem x aus den bekannten Daten berechnet werden kann.

Für $k = 2$ ist dies leicht möglich: Da m_1 und m_2 teilerfremd sind, kann man mit dem Euklidischen Algorithmus ganze Zahlen u und v berechnen, so daß $um_1 + vm_2 = 1$ ist. Offensichtlich erfüllt dann

$$x \equiv um_1x_2 + vm_2x_1 \bmod m_1m_2$$

die Kongruenzen $x \equiv x_i \bmod m_i$ $(i = 1, 2)$. Durch iterative Anwendung dieser Beobachtung ergibt sich der folgende Algorithmus (vgl. [13], S. 17).

Algorithmus 3.34 (Chinesischer Restsatz)

Eingabe:	$m_1, \dots, m_k \in \mathbb{N}$ teilerfremd und $x_1, \dots, x_k \in \mathbb{Z}$
Ausgabe:	die eindeutig bestimmte simultane Lösung x der Kongruenzen (3.2), die die Abschätzung (3.1) erfüllt

Setze $m \leftarrow m_1$ und $x \leftarrow x_1 \bmod m_1$.		(1)
FOR $i = 1$ TO k		(2)
	Berechne mit dem Euklidischen Algorithmus die Koeffizienten u, v aus der Gleichung $um + vm_i = 1$.	(3)
	Setze $x \leftarrow umx_i + vm_ix$ und dann $m \leftarrow mm_i$, $x \leftarrow x \bmod m$.	(4)

Dieser Algorithmus kann nun in der Praxis zur Vereinfachung von Berechnungen in $\mathbb{Z}$ ausgenutzt werden, bei denen große Zwischenergebnisse auftreten. Man bestimmt dazu zunächst eine obere Schranke M für die Größe der zu berechnenden Zahl x. Dann bestimmt man die Reste x_i von x modulo einiger Primzahlen p_i, die so zu wählen sind, daß zum einen ihr Produkt M übersteigt und zum anderen die Arithmetik modulo p_i effizient ist. Zuletzt rekonstruiert man x mit Hilfe von Algorithmus 3.34.

Für die Anwendung des Chinesischen Restsatzes zur Berechnung der Determinante einer Matrix $A \in \mathbb{Z}^{n \times n}$ ist es also notwendig, im voraus eine obere Abschätzung für $|\det A|$ zu kennen. Diese erhalten wir mit Hilfe der Hadamardschen Ungleichung (vgl. [23], S. 414)

$$|\det A| \leq \prod_{j=1}^{n} \left(\sum_{i=1}^{n} |a_{ij}|^2 \right)^{1/2}. \tag{3.3}$$

Kennt man eine obere Schranke $\overline{a}$ für die Matrixeinträge, d.h. eine Zahl $\overline{a}$ mit

$$\overline{a} \geq \max\{|a_{ij}| \ : \ 1 \leq i, j \leq n\},$$

so folgt aus der Hadamardschen Ungleichung unmittelbar

$$|\det A| \leq (\sqrt{n}\,\overline{a})^n. \tag{3.4}$$

Insgesamt ergibt sich der folgende Algorithmus für die Berechnung der Determinante einer ganzzahligen Matrix:

Algorithmus 3.35 (Modulare Gauß-Elimination)

Eingabe: $A \in \mathbb{Z}^{n \times n}$
Ausgabe: $d = \det A \in \mathbb{Z}$

Bestimme mit der Hadamardschen Ungleichung eine obere Schranke $\tilde{d}$ für $\|d\|$.		(1)
Bestimme eine Menge $\{p_1, \ldots, p_k\}$ von Primzahlen mit $p = \prod_{i=1}^{k} p_i > 2\tilde{d}$.		(2)
FOR $i = 1$ TO k		(3)
	Berechne $d_i \in [\![0, p_i - 1]\!]$ mit $d_i \equiv \det A \bmod p_i$. Fasse dazu A als Matrix über dem Körper $K = (\mathbb{Z}/p_i\mathbb{Z})^*$ auf und berechne mit Gauß-Elimination die Determinante von A über K.	(4)
Berechne mit dem Chinesischen Restsatz die eindeutig bestimmte Zahl $d \in [\![0, p-1]\!]$, die die Kongruenzen $d \equiv d_i \bmod p_i$ $(1 \leq i \leq k)$ erfüllt.		(5)
IF $d \geq \tilde{d}$ THEN setze $d \leftarrow d - p$.		(6)

Kapitel 4

Die Grundidee für die Klassengruppenberechnung

Ziel dieses Kapitels ist es, die Grundidee zu erläutern, auf der sowohl der Algorithmus von Hafner und McCurley als auch der von uns entwickelte Algorithmus CLASSGROUP beruhen.

In den ersten beiden Abschnitten werden zunächst einige spezielle Resultate aus der Literatur dargestellt, die im Zusammenhang mit der Klassengruppenberechnung benötigt werden. Im einzelnen geht es dabei um die Konstruktion eines Erzeugendensystems für die Klassengruppe und um die Berechnung einer geeigneten Approximation für die Klassenzahl.

Im dritten Abschnitt wird dann die grundsätzliche Idee der Berechnung von Klassengruppen mit Hilfe sogenannter Relationen erläutert. Im vierten Abschnitt wird die Grundidee zur Generierung dieser Relationen dargestellt.

Im fünften und sechsten Abschnitt werden schließlich zwei Algorithmen aus der elementaren Zahlentheorie angegeben, die für die Berechnung des Erzeugendensystems und für die Approximation der Klassenzahl benötigt werden.

Einige der Resultate in den ersten beiden Abschnitten hängen von einer unbewiesenen Annahme ab, der sogenannten **Verallgemeinerten Riemannschen Vermutung**. Sie können nur bewiesen werden, wenn man diese Vermutung als wahr annimmt. Solche Sätze sind mit dem Zusatz (GRH) gekennzeichnet.

Die gewöhnliche Riemannsche Vermutung ist die Annahme, daß alle komplexen Nullstellen der sogenannten "Riemannschen Zeta-Funktion" $\zeta(x)$, deren Realteil zwischen 0 und 1 liegt, einen Realteil von genau $\frac{1}{2}$ haben. Die Verallgemeinerte Riemannsche Vermutung ist dieselbe Annahme für bestimmte Verallgemeinerungen der Funktion $\zeta(x)$, die sogenannten "Dirichletschen L-Reihen".

Die Verallgemeinerte Riemannsche Vermutung spielt auch in vielen anderen Gebieten der Zahlentheorie eine große Rolle. Für weitere Einzelheiten verweisen wir auf [25], S. 55/56.

4.1 Konstruktion eines Erzeugendensystems

In diesem Abschnitt wird erläutert, wie man mit Hilfe sogenannter Primformen ein Erzeugendensystem für die Klassengruppe konstruieren kann und wie Elemente der Klassengruppe über diesem Erzeugendensystem zerlegt werden können. Alle Aussagen dieses Abschnittes gelten für positive und negative Diskriminanten.

Wir definieren zunächst den Begriff des quadratischen Restes.

Definition 4.1 *Es sei $q \in \mathbb{N}$. Eine Zahl $a \in \mathbb{Z}$ heißt* **quadratischer Rest modulo q**, *wenn es eine Zahl $b \in \mathbb{Z}$ gibt, so daß $b^2 \equiv a \bmod q$ ist.*

Wir fixieren nun eine Diskriminante $\Delta \in \mathcal{D}$ und untersuchen die Frage, für welche Primzahlen p es Formen (p, b) der Diskriminante Δ gibt. Anhand von Definition 2.1 stellt man leicht fest, daß solche (nicht notwendigerweise primitiven) Formen genau dann existieren, wenn Δ quadratischer Rest modulo $(4p)$ ist. Um diese Bedingung kürzer zu schreiben, führt man das sogenannte Kroneckersymbol ein.

Definition 4.2 *Es sei $\Delta \in \mathcal{D}$, $p \in \mathbb{P}$. Das* **Kroneckersymbol** $(\frac{\Delta}{p})$ *ist definiert durch*

$$\left(\frac{\Delta}{p}\right) = \begin{cases} 1 & \text{falls } \Delta \text{ quadratischer Rest modulo } 4p \text{ ist und } p \nmid \Delta, \\ 0 & \text{falls } p \mid \Delta, \\ -1 & \text{sonst.} \end{cases} \tag{4.1}$$

Formen (p, b) der Diskriminante Δ existieren somit genau für diejenigen Primzahlen p, für die das Kroneckersymbol $(\frac{\Delta}{p})$ den Wert 0 oder 1 hat. Natürlich gibt es in diesem Fall unendlich viele Formen (p, b) der Diskriminante Δ, denn mit (p, b) liegt auch $(p, b + 2p)$ in $\mathcal{QF}(\Delta)$. Beschränkt man sich auf Primzahlen p mit $(\frac{\Delta}{p}) = 1$, schließt also Primzahlen aus, die die Diskriminante teilen, so sind alle diese Formen primitiv.

Definition 4.3 *Es sei $p \in \mathbb{P}$ mit $(\frac{\Delta}{p}) = 1$ und*

$$b_p = \min\{b \in \mathbb{N} \ : \ b^2 \equiv \Delta \bmod (4p)\}. \tag{4.2}$$

Die Form $f_p = (p, b_p)$ heißt **Primform**.

Mit Hilfe von Primformen kann man ein Erzeugendensystem für die Klassengruppe konstruieren. Dazu wird in [37] der folgende Satz bewiesen:

Satz 4.4 *(GRH) Es sei $\Delta \in \mathcal{D}$. Dann existiert eine absolute, effektiv berechenbare Konstante $c_1 \in \mathbb{R}_{>0}$, so daß die Menge*

$$\left\{[f_p] \ : \ f_p \text{ Primform in } \mathcal{QF}(\Delta),\ (\tfrac{\Delta}{p}) = 1,\ p < c_1 \log^2 |\Delta|\right\}$$

ein Erzeugendensystem der Klassengruppe $Cl(\Delta)$ ist.

In [2] wird gezeigt, daß man für negative Diskriminante $c_1 = 6$, für positive Diskriminante $c_1 = 12$ wählen kann.

Der folgende Satz gibt an, wie man beliebige Elemente aus der Klassengruppe in ein Potenzprodukt anderer Klassen zerlegen kann (Beweis in [34], vgl. auch [38]).

Satz 4.5 *Es sei* $(a, b) \in \mathcal{QF}(\Delta)$ *mit* $\gcd(a, \Delta) = 1$. *Es sei*

$$a = \prod_{i=1}^{n} p_i^{e_i}$$

mit geeignetem $n \in \mathbb{N}$ *sowie* $e_1, \ldots, e_n \in \mathbb{N}$ *und* $p_1, \ldots, p_n \in \mathbb{P}$ *die Primfaktorzerlegung von* a. *Dann gilt:*

(i)
$$\left(\tfrac{\Delta}{p_i}\right) = 1 \qquad (1 \leq i \leq n).$$

(ii)
$$b \equiv \pm b_{p_i} \; mod\,(2p_i) \qquad (1 \leq i \leq n),$$

wobei b_{p_i} *jeweils gemäß (4.2) durch* p_i *definiert ist.*

(iii)
$$[(a, b)] = \prod_{i=1}^{n} [f_{p_i}]^{\pm e_i}, \tag{4.3}$$

wobei das positive Vorzeichen des Exponenten e_i *jeweils genau dann gilt, wenn* $b \equiv +b_{p_i} \; mod\,(2p_i)$ *ist.*

Mit Hilfe dieses Satzes kann also eine Klasse $[f] \in Cl(\Delta)$, die durch eine beliebige Form $f = (a, b) \in \mathcal{QF}(\Delta)$ mit $\gcd(a, \Delta) = 1$ gegeben ist, in ein Potenzprodukt solcher Klassen zerlegt werden, die Primformen enthalten. Zur Anwendung dieses Satzes muß die Primfaktorzerlegung von a bekannt sein. Wir benötigen diesen Satz später, um Formenklassen über einem nach Satz 4.4 konstruierten Erzeugendensystem der Klassengruppe zu zerlegen. Obwohl es sich hier formal um eine Zerlegung von Klassen und nicht von Formen handelt, bezeichnen wir diesen Vorgang auch kurz als die **Zerlegung von f in Primformen**.

4.2 Approximation der Klassenzahl

Wir beschränken uns nun wieder auf negative Diskriminanten und stellen einige Aussagen über die Größe der Klassenzahl zusammen.

Wir definieren zunächst für $\Delta \in \mathcal{D}^-$ und $p \in \mathbb{P}$ die **Eulerfaktoren**

$$E(\Delta, p) = \left(1 - \left(\frac{\Delta}{p}\right) \cdot \frac{1}{p}\right)^{-1}.$$

Für imaginär-quadratische Klassenzahlen erhält man dann mit Hilfe der analytischen Klassenzahlformel folgende Aussage (vgl. [37]):

Satz 4.6 *Es sei $\Delta \in \mathcal{D}^-$. Dann ist*

$$h(\Delta) = \frac{\sqrt{|\Delta|}}{\pi} \cdot \prod_{p \in \mathbb{P}} E(\Delta, p). \tag{4.4}$$

Allerdings konvergiert dieses unendliche Produkt nur sehr langsam, so daß es nicht möglich ist, die Klassenzahl für große Diskriminanten auf diese Weise effizient zu berechnen. Man kann jedoch mit Hilfe der Auswertung endlich vieler Faktoren von (4.4) eine Approximation

$$\tilde{h}(\Delta, Q) = \frac{\sqrt{|\Delta|}}{\pi} \prod_{\substack{p \in \mathbb{P}, \\ p \leq Q}} E(\Delta, p) \tag{4.5}$$

an die Klassenzahl berechnen ($Q \in \mathbb{N}$). Der folgende Satz zeigt, daß man dabei die Größe des Fehlers beschränken kann (Beweis ebenfalls in [37]).

Satz 4.7 *(GRH) Es sei $\Delta \in \mathcal{D}^-$. Dann existieren absolute, effektiv berechenbare Konstanten $c_2, c_3 \in \mathbb{R}_{>0}$, so daß für alle $Q > c_2 \log^2 |\Delta|$ die Abschätzung*

$$\left| 1 - \prod_{\substack{p \in \mathbb{P}, \\ p > Q}} E(\Delta, p)^{-1} \right| < \frac{c_3 \log |\Delta Q|}{\sqrt{Q}} \tag{4.6}$$

gilt.

Aus diesem Satz folgt unmittelbar (vgl. [29]):

Korollar 4.8 *(GRH) Es sei $\Delta \in \mathcal{D}^-$. Dann existiert eine absolute, effektiv berechenbare Konstante $c_4 \in \mathbb{R}_{>0}$, so daß für alle $Q > c_4 \log^2 |\Delta|$ die Abschätzung*

$$\frac{3}{4} < \frac{h(\Delta)}{\tilde{h}(\Delta, Q)} < \frac{3}{2} \tag{4.7}$$

gilt.

Wir geben hier noch einige weitere Aussagen über die Größe der Klassenzahl in Abhängigkeit von der Diskriminante an (Beweise in [30], S. 389).

Zunächst gilt folgende obere Abschätzung:

$$h(\Delta) \leq \frac{2\sqrt{|\Delta|}}{\pi} \left(1 + \log \left(\frac{2\sqrt{|\Delta|}}{\pi} \right) \right).$$

Für $|\Delta| > e^{24}$ ergibt sich daraus die einfachere Formel

$$h(\Delta) \leq \tfrac{1}{3} \cdot \sqrt{|\Delta|} \cdot \log |\Delta|.$$

Für genügend großes $|\Delta|$ gilt zudem für jedes $\varepsilon > 0$ die untere Abschätzung

$$h(\Delta) \geq |\Delta|^{1/2+\varepsilon}.$$

Beide Abschätzungen gemeinsam lassen den Schluß zu, daß die Klassenzahl $h(\Delta)$ in etwa die Größenordnung $\sqrt{|\Delta|}$ hat.

4.3 Klassengruppenberechnung mit Hilfe von Relationen

Wir beschreiben in diesem Abschnitt die grundsätzliche Idee zur Berechnung der Klassengruppe, auf der sowohl der Algorithmus von Hafner und McCurley als auch der von uns entwickelte Algorithmus CLASSGROUP beruhen. Diese Idee wurde bereits vor ihrer Anwendung in [18] in verschiedenen anderen Arbeiten benutzt, z.B. auch in [29]. Die Darstellung wird hier bewußt sehr allgemein vorgenommen, um die Grundidee herauszustellen und von speziellen Realisierungen zu abstrahieren.

Es sei eine Diskriminante $\Delta \in \mathcal{D}^-$ fest vorgegeben. Wie in Kapitel 2 sei $\mathcal{QF}(\Delta)$ die Menge der positiv definiten binären quadratischen Formen der Diskriminante Δ, $Cl = Cl(\Delta)$ die zugehörige Klassengruppe, $h = h(\Delta)$ die Klassenzahl und 1_{Cl} das neutrale Element in der Klassengruppe.

Wir gehen aus von einer Menge von Formen

$$F = \{f_1, \ldots, f_n\} \subset \mathcal{QF}(\Delta),$$

die so gewählt ist, daß die Menge

$$\tilde{F} = \{[f_1], \ldots, [f_n]\}$$

der zugehörigen Äquivalenzklassen die Klassengruppe erzeugt. Wir bezeichnen eine Menge F mit dieser Eigenschaft als **Formenbasis** für die Klassengruppe.

Eine Formenbasis F kann nach Satz 4.4 leicht berechnet werden, indem man einen Parameter $N > c_1 \log^2 D$ wählt (c_1 wie in Satz 4.4) und

$$F = \{f_p = (p, b_p) \ : \ f_p \text{ Primform in } \mathcal{QF}(\Delta),\ p < N\} \tag{4.8}$$

setzt. Eine so konstruierte Formenbasis enthält alle Primformen $(p, b_p) \in \mathcal{QF}(\Delta)$, für die p unterhalb der gewählten Schranke N liegt. Diese zusätzliche Eigenschaft der Formenbasis ist später von Bedeutung, spielt aber für die Betrachtungen in diesem Abschnitt zunächst keine Rolle.

Wir definieren nun die Abbildung

$$\begin{array}{rccl} \varphi: & \mathbb{Z}^n & \to & Cl \\ & (e_1, \ldots, e_n) & \mapsto & \prod_{i=1}^{n} [f_i]^{e_i}. \end{array}$$

Offensichtlich ist φ ein Homomorphismus und, da $\tilde{F}$ die Klassengruppe erzeugt, ist φ auch surjektiv. Der Kern der Abbildung φ ist

$$\Lambda = \ker \varphi = \left\{ (e_1, \ldots, e_n) \ : \ \prod_{i=1}^{n} [f_i]^{e_i} = 1_{Cl} \right\}.$$

Offenbar ist Λ ein Gitter im $\mathbb{Z}^n$. Nach dem Homomorphiesatz für Gruppen (vgl. z.B. [43]) ist außerdem

$$\mathbb{Z}^n / \Lambda \cong Cl.$$

Da $h = \#Cl$ endlich ist, hat Λ endlichen Index in $\mathbb{Z}^n$, d.h. $\dim \Lambda = n$, und es gilt $\mathrm{Det}\ \Lambda = h$. Wir können also die Klassenzahl h berechnen, indem wir die Determinante des Gitters Λ bestimmen.

Wir bezeichnen die Vektoren in Λ als **Relationen** (über F), das Gitter Λ selbst als **Relationengitter** (über F).

Die grundsätzliche Idee zur Klassengruppenberechnung besteht nun darin, zunächst mit Hilfe eines probabilistischen Verfahrens, auf das wir im nächsten Abschnitt genauer eingehen, eine gewisse Anzahl von Relationen über F zu bestimmen und dann mit Hilfe der Algorithmen aus Kapitel 3 die Determinante d des von diesen Relationen erzeugten Gitters zu berechnen. Wenn die Relationen das volle Relationengitter Λ erzeugen, ist nach den obigen Betrachtungen die Klassenzahl h gleich der Determinante d. Außerdem können wir in diesem Fall, ebenfalls mit den Algorithmen aus Kapitel 3, die Struktur der Klassengruppe Cl berechnen.

Um zu entscheiden, ob die Menge R der gefundenen Relationen das volle Relationengitter Λ erzeugt, benutzen wir eine Approximation h^* für h mit

$$h^* < h < 2h^*. \tag{4.9}$$

Eine solche Approximation kann nach Korollar 4.8 bestimmt werden, indem man $\tilde{h}(\Delta, Q)$ gemäß Gleichung (4.5) für $Q > c_4 \log^2 |\Delta|$ berechnet (c_4 wie in Korollar 4.8) und dann $h^* = \frac{3}{4}\tilde{h}(\Delta, Q)$ setzt.

Da alle Relationen im Gitter Λ liegen, erzeugt die Menge R ein Teilgitter Λ' von Λ. Falls $\dim \Lambda' = n$ ist, gilt nach Satz 3.7

$$h = \mathrm{Det}\ \Lambda \mid \mathrm{Det}\ \Lambda'.$$

Gilt in diesem Fall außerdem $\mathrm{Det}\ \Lambda' < 2h^*$, so folgt unmittelbar $\mathrm{Det}\ \Lambda' = \mathrm{Det}\ \Lambda = h$ und $\Lambda' = \Lambda$.

Zur Formalisierung der Ergebnisse dieses Abschnittes führen wir nun noch den Begriff des Relationensystems ein:

Definition 4.9 *Es sei $\Delta \in \mathcal{D}^-$. Ein Tupel $\mathcal{R} = (F, A, n, m)$ mit $n, m \in \mathbb{N}$, $m \geq n$, $F = \{f_1, \ldots, f_n\} \subset \mathcal{QF}(\Delta)$ und $A \in \mathbb{Z}^{n \times m}$ heißt* **Relationensystem** *für die Klassengruppe $Cl(\Delta)$, wenn die Menge F eine Formenbasis für die Klassengruppe $Cl(\Delta)$ ist und die Spalten der Matrix A Relationen über F sind, d.h.*

$$\prod_{i=1}^{n} [f_i]^{a_{ij}} = 1_{Cl} \quad (1 \leq j \leq m). \tag{4.10}$$

Mit den Überlegungen in diesem Abschnitt haben wir den folgenden Satz bewiesen:

Satz 4.10 *Es sei $\Delta \in \mathcal{D}^-$ und $\mathcal{R} = (F, A, n, m)$ ein Relationensystem für $Cl(\Delta)$. Es sei h^* eine Approximation für $h(\Delta)$ mit $h^* < h(\Delta) < 2h^*$. Dann gilt: Ist $\mathrm{rang}\, A = n$ und $d = \mathrm{Det}\, \Lambda(A) < 2h^*$, so folgt $h(\Delta) = d$ und $\mathbb{Z}^n / \Lambda(A) \cong Cl(\Delta)$.*

Die in diesem Abschnitt beschriebene grundsätzliche Idee zur Berechnung der Ordnung und der Struktur der Klassengruppe läßt sich von der Theorie her unmittelbar auf beliebige andere endliche abelsche Gruppen übertragen. Man benötigt dazu allerdings zum einen Aussagen darüber, wie man ein Erzeugendensystem für die jeweilige Gruppe G konstruieren und eine Approximation g^* mit $g^* < \#G < 2g^*$ berechnen kann, zum anderen ist ein Verfahren zum Auffinden von Relationen in der Gruppe G erforderlich. Insbesondere letzteres steht jedoch für beliebige endliche abelsche Gruppen nicht zur Verfügung. Das Verfahren zur Generierung der Relationen, das im nächsten Abschnitt beschrieben wird, nutzt spezielle Eigenschaften der Klassengruppe aus und läßt sich nur auf sehr wenige andere Gruppen übertragen.

4.4 Die Generierung der Relationen

Wir erläutern in diesem Abschnitt die grundsätzliche Idee zur Generierung von Relationen in der Klassengruppe. Diese Idee ist in einer Arbeit von Seysen [38] enthalten und wurde dort zur Konstruktion eines Faktorisierungsalgorithmus verwendet.

Wir setzen voraus, daß die Formenbasis F, über der die Relationen zu konstruieren sind, die Form (4.8) hat, also alle Primformen $f_p = (p, b_p) \in \mathcal{QF}(\Delta)$ enthält, für die p unterhalb einer gewissen Schranke N liegt. Die Zahl n bezeichnet wiederum die Anzahl der Elemente in F. Wir bezeichnen ferner im folgenden mit F_p die Menge der Primzahlen, die zu den Primformen in F gehören, also

$$F_p = \{p \in \mathbb{P} \ : \ f_p \in F\}.$$

Zur Generierung einer Relation wählt man zunächst einen Vektor $x \in \mathbb{Z}^n$ nach einem noch festzulegenden Verfahren, z.B. durch zufällige Auswahl der Vektoreinträge $x_1, \ldots, x_n$ aus einer beschränkten Teilmenge von $\mathbb{Z}$. Man berechnet dann die reduzierte Form $g = (a, b) \in \mathcal{QF}(\Delta)$ in der Klasse des Potenzproduktes

$$f = (a', b') = \prod_{i=1}^{n} f_i^{x_i}$$

und versucht, eine Zerlegung der Form g über der Formenbasis $F = \{f_1, \ldots, f_n\}$ zu finden. Wenn das gelingt, d.h.

$$g \sim \prod_{i=1}^{n} f_i^{y_i},$$

so gilt

$$1_{\mathcal{Cl}} = [g \cdot g^{-1}] = [f \cdot g^{-1}] = \prod_{i=1}^{n} [f_i]^{x_i - y_i},$$

der Vektor $e = (e_1, \ldots, e_n) = (x_1 - y_1, \ldots, x_n - y_n)$ ist also eine Relation. Ist die Form g über der Formenbasis nicht zerlegbar, so wählt man einen anderen Vektor x und beginnt von vorn.

Die Zerlegung der Form $g = (a, b)$ über der Formenbasis F erfolgt gemäß Satz 4.5. Dazu muß man zunächst feststellen, ob $\gcd(a, \Delta) = 1$ ist und ob a vollständig in

Primfaktoren $p < N$ zerfällt. Da F alle Primformen $f_p = (p, b_p)$ mit $p \leq N$ enthält, ist g in diesem Fall vollständig über F zerlegbar. Zu jedem Primfaktor p von a existiert dann nämlich nach Satz 4.5 eine Primform $f_p = (p, b_p) \in F$, die in der Zerlegung von g auftritt. Hat p in der Zerlegung von a den Exponenten e_p, so hat f_p in der Zerlegung von g den Exponenten $e'_p = \pm e_p$. Das korrekte Vorzeichen von e'_p kann ebenfalls nach Satz 4.5 bestimmt werden. Für die Zerlegung der Form g ist es also von Bedeutung, daß die Formenbasis F gemäß (4.8) konstruiert ist.

Für die Erzeugung der Relationen ergibt sich somit der folgende Algorithmus:

Algorithmus 4.11 (Seysen)

Eingabe: die Diskriminante Δ, die Formenbasis F und $n = \#F$
Ausgabe: eine Relation $w = (w_1, \ldots, w_n)$ über F

	Wähle einen Vektor $(x_1, \ldots, x_n) \in \mathbb{Z}^n$.	(1)
	Berechne die reduzierte Form $g = (a, b)$ in der Klasse von $\prod_{i=1}^{n} f_i^{x_i}$.	(2)
	Versuche, eine Zerlegung $a = \prod_{i=1}^{n} p_i^{u_i}$ von a über F_p zu finden.	(3)
UNTIL die vollständige Zerlegung von a ist möglich.		(4)
FOR $i = 1$ TO n berechne $y_i = \pm u_i$, so daß $g \sim \prod_{i=1}^{n} f_i^{y_i}$.		(5)
FOR $i = 1$ TO n setze $w_i \leftarrow x_i - y_i$.		(6)

Wir bezeichnen im folgenden den Vektor x, der die für die Bildung des Potenzproduktes gewählten Exponenten enthält, als **Exponentenvektor**. Ferner bezeichnen wir den Vektor u, der die Exponenten der Primzahlen aus F_p in der Zerlegung von a enthält, als **Zerlegungsvektor** und den Vektor y, der die Exponenten der Primformen aus der Formenbasis in der Zerlegung der Form g enthält, als **Formenzerlegungsvektor**. Die Einträge des Zerlegungsvektors und des Formenzerlegungsvektors unterscheiden sich höchstens im Vorzeichen.

4.5 Berechnung des Kroneckersymbols

Sowohl für die Konstruktion des Erzeugendensystems für die Klassengruppe nach Satz 4.4 als auch für die Berechnung der Approximation h^* mit Gleichung (4.5) muß man den Wert des Kroneckersymbols $(\frac{\Delta}{p})$ für eine Reihe von Primzahlen p bestimmen. Wir geben dazu in diesem Abschnitt einen Algorithmus an.

Wir untersuchen zunächst den Fall $p = 2$. Hier hängt das Kroneckersymbol, wie man leicht verifizieren kann, nur vom Wert von Δ modulo 8 ab. Es ist

$$\left(\frac{\Delta}{p}\right) = \begin{cases} 1 & \text{falls } \Delta \equiv 1 \bmod 8, \\ 0 & \text{falls } \Delta \equiv 0 \bmod 4, \\ -1 & \text{falls } \Delta \equiv 5 \bmod 8. \end{cases} \tag{4.11}$$

Dies sind alle möglichen Fälle, da das Kroneckersymbol nur für $\Delta \equiv 0, 1 \bmod 4$ definiert ist.

Für Primzahlen $p > 2$ ist der Wert des Kroneckersymbols nicht auf so einfache Weise zu ermitteln. Hier sind noch einige theoretische Betrachtungen erforderlich.

Wir geben zunächst den Zusammenhang zwischen quadratischen Resten modulo p und modulo $(4p)$ an. Die Aussage des folgenden Satzes kann man leicht verifizieren, indem man die einzelnen Fälle nachrechnet.

Satz 4.12 *Es sei $\Delta \equiv 0, 1 \bmod 4$ und $p \in \mathbb{P},\ p \neq 2$. Es sei $b \in \mathbb{Z}$ mit*

$$b^2 \equiv \Delta \bmod p.$$

Dann folgt

$$b'^2 \equiv \Delta \bmod (4p) \tag{4.12}$$

für

$$b' = \begin{cases} b & \text{falls } b \equiv \Delta \bmod 2, \\ p - b & \text{sonst.} \end{cases} \tag{4.13}$$

Weiterhin kann man leicht verifizieren, daß die nach Satz 4.12 berechnete Lösung b' die kleinste (nicht negative) Lösung der Kongruenz (4.12) ist, wenn $0 \leq b < p$ war.

Aus Satz 4.12 folgt unmittelbar:

Korollar 4.13 *Es sei $\Delta \equiv 0, 1 \bmod 4$ und $p \in \mathbb{P},\ p \neq 2$. Δ ist genau dann quadratischer Rest modulo $(4p)$, wenn Δ quadratischer Rest modulo p ist.*

Man beachte, daß die Aussage von Korollar 4.13 nur unter den angegebenen Voraussetzungen gilt. Sie ist nicht auf allgemeines Δ übertragbar.

Um die folgenden Sachverhalte in der allgemein üblichen Weise darstellen zu können, definieren wir nun das Legendresymbol.

Definition 4.14 *Es sei $a \in \mathbb{Z}$ und $p \in \mathbb{P},\ p \neq 2$. Das* **Legendresymbol** $(\frac{a}{p})$ *ist definiert durch*

$$\left(\frac{a}{p}\right) = \begin{cases} 1 & \text{falls } a \text{ quadratischer Rest modulo } p \text{ und } p \nmid a, \\ 0 & \text{falls } p \mid a, \\ -1 & \text{sonst.} \end{cases} \tag{4.14}$$

Man beachte, daß Legendresymbol und Kroneckersymbol unterschiedliche Definitionsbereiche für Zähler und Nenner haben. In den Fällen, in denen beide Symbole definiert sind, stimmen sie jedoch überein, so daß die gleiche Bezeichnungsweise gerechtfertigt ist. Zur Berechnung des Legendresymbols kann man den folgenden Satz verwenden (Beweis z.B. in [20], S. 36):

Satz 4.15 (Eulersches Kriterium) *Es sei $a \in \mathbb{Z}$ und $p \in \mathbb{P}, p \neq 2$. Dann gilt*

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

Die Berechnung des Legendresymbols mit Hilfe des Eulerschen Kriteriums ist in der Praxis für große Werte von a oder p nicht effizient, da dann viele Multiplikationen und Reduktionen großer Zahlen durchgeführt werden müssen. Wir verwenden daher einen anderen Algorithmus, der sich in der Praxis als effizienter erwiesen hat. Zur Formulierung dieses Algorithmus wird der Begriff des Jacobisymbols benötigt, der eine Verallgemeinerung des Legendresymbols darstellt. Wir definieren den Begriff des Jacobisymbols hier nur für ungerade positive Nenner, da dieser Fall für unsere Zwecke ausreichend ist.

Definition 4.16 *Es seien $a \in \mathbb{Z}$ und $n \in \mathbb{N}$ ungerade mit $n \neq 1$. Es sei*

$$n = \prod_{i=1}^{k} p_i^{e_i}$$

mit geeignetem $k \in \mathbb{N}$ sowie $p_1, \ldots, p_k \in \mathbb{P}$ und $e_1, \ldots, e_k \in \mathbb{N}$ die Primfaktorzerlegung von n. Das **Jacobisymbol** $\left(\frac{a}{n}\right)$ *ist definiert durch*

$$\left(\frac{a}{n}\right) = \prod_{i=1}^{k} \left(\frac{a}{p_i}\right)^{e_i}, \tag{4.15}$$

wobei $\left(\frac{a}{p_i}\right)$ die enstprechenden Legendresymbole bezeichnet.

Das Jacobisymbol hat unter anderem die im folgenden Satz zusammengestellten Eigenschaften (Beweise z.B. in [20], S. 45/46):

Satz 4.17 *Es seien $a \in \mathbb{Z}$ und $n, m \in \mathbb{N}$ ungerade mit $n \neq 1$, $m \neq 1$. Dann gilt:*

$$\left(\frac{a}{n}\right) = \left(\frac{a+kn}{n}\right) \quad \text{für alle } k \in \mathbb{Z}, \tag{4.16}$$

$$\left(\frac{2}{n}\right) = (-1)^{(n^2-1)/8}, \tag{4.17}$$

$$\left(\frac{1}{n}\right) = 1, \tag{4.18}$$

$$\left(\frac{0}{n}\right) = 0, \tag{4.19}$$

$$\left(\frac{m}{n}\right) = (-1)^{\frac{(n-1)(m-1)}{4}} \left(\frac{n}{m}\right). \tag{4.20}$$

Die letzte Gleichung wird als das **quadratische Reziprozitätsgesetz** bezeichnet.

Mit Hilfe dieser Regeln läßt sich ein effizienter Algorithmus zur Berechnung des Jacobisymbols formulieren:

Algorithmus 4.18 (Jacobisymbol)

Eingabe: $a \in \mathbb{Z}$, $n \in \mathbb{N}$ ungerade, $n \neq 1$
Ausgabe: der Wert J des Jacobisymbols $\left(\frac{a}{n}\right)$

Setze $J \leftarrow 1$.				(1)
	Reduziere a modulo n, so daß $0 \leq a < n$.			(2)
	IF $a = 0$			(3)
	THEN	Setze $J \leftarrow 0$.		(4)
	ELSE	Setze $l \leftarrow 0$.		(5)
		WHILE a ist gerade		(6)
			Setze $a \leftarrow a/2$ und $l \leftarrow l + 1$.	(7)
		IF l ist ungerade THEN setze $J \leftarrow (-1)^{(n^2-1)/8} \cdot J$.		(8)
BREAKIF $a = 0$ oder $a = 1$.				(9)
	Setze $J \leftarrow (-1)^{\frac{(a-1)(n-1)}{4}} \cdot J$ und vertausche a und n.			(10)

Die Korrektheit des Algorithmus ergibt sich aus den Formeln in Satz 4.17. Die in den Schritten (8) und (10) auftretenden Exponenten müssen in der Praxis nicht explizit berechnet werden, da es nur auf ihren Wert modulo 2 ankommt. Der Algorithmus terminiert nach höchstens $(\log n + 1)$ Durchläufen der Schleife (2) - (10), da der Wert von n in jedem Durchlauf dieser Schleife (mit Ausnahme des ersten) mindestens um einen Faktor 2 vermindert wird.

4.6 Berechnung von Primformen

Wir geben in diesem Abschnitt einen Algorithmus zur Berechnung von Primformen an. Ein solcher Algorithmus wird zur Konstruktion eines Erzeugendensystems für die Klassengruppe nach Satz 4.4 benötigt. Die Aufgabe des Algorithmus besteht darin, für eine vorgegebene Primzahl p, von der man schon festgestellt hat, daß $\left(\frac{\Delta}{p}\right) = 1$ ist, die durch (4.2) bestimmte Zahl $b_p \in \mathbb{N}$ zu berechnen, so daß (p, b_p) Primform in $\mathcal{QF}(\Delta)$ ist.

Für $p = 2$ ist die Bestimmung von b_p einfach. Für $\left(\frac{\Delta}{2}\right) = 1$ folgt nach (4.11), daß $\Delta \equiv 1 \bmod 8$ ist, und daher ist offensichtlich $b_2 = 1$.

Für $p > 2$ mit $\left(\frac{\Delta}{p}\right) = 1$ bestimmen wir zunächst eine Lösung b' der Kongruenz

$$x^2 \equiv \Delta \bmod p. \tag{4.21}$$

Dann können wir nach Satz 4.12 eine Lösung b der Kongruenz $x^2 \equiv \Delta \bmod (4p)$ berechnen. Diese Lösung b ist zudem die kleinste (nicht negative) Lösung dieser Kongruenz, es ist also $b_p = b$.

Für $p \equiv 3 \bmod 4$ kann man die Lösung der Kongruenz (4.21) mit Hilfe des folgenden Satzes für etwa die Hälfte der Primzahlen p, nämlich für diejenigen mit $p \equiv 3 \bmod 4$, finden. Der Beweis ergibt sich unmittelbar aus Satz 4.15.

Satz 4.19 *Es sei $a \in \mathbb{Z}$ und $p \in \mathbb{P}$, $p \equiv 3 \bmod 4$ mit $(\frac{a}{p}) = 1$. Dann ist*

$$b \equiv a^{(p+1)/4} \bmod p \tag{4.22}$$

eine Lösung der Kongruenz (4.21).

Für $p \equiv 1 \bmod 4$ ist die Lösung der Kongruenz (4.21) nicht auf so einfache Weise zu bestimmen. Wir verwenden in diesem Fall den Algorithmus **RESSOL**, der von Shanks [40] entwickelt wurde. Dieser Algorithmus löst die Kongruenz (4.21) für beliebige ungerade Primzahlen, ist also nicht auf den Fall $p \equiv 1 \bmod 4$ beschränkt. Wir erläutern zunächst die Idee des Algorithmus.

Dazu sei p eine ungerade Primzahl und $a \in \mathbb{Z}$ mit $(\frac{a}{p}) = 1$, d.h. insbesondere $\gcd(a,p) = 1$. Ziel des Algorithmus ist es, eine Zahl $r \in \mathbb{Z}$ zu berechnen mit $r^2 \equiv a \bmod p$. Wir bezeichnen ferner für $x \in \mathbb{Z}$ die Restklasse von x modulo p, aufgefaßt als Elemente der multiplikativen Gruppe $(\mathbb{Z}/p\mathbb{Z})^*$, mit $\overline{x}$.

Die Idee von Shanks besteht darin, eine Folge $(r_i, n_i) \in \mathbb{Z}^2$ zu berechnen, so daß

$$r_i^2 \equiv a \cdot n_i \bmod p \tag{4.23}$$

gilt und die Ordnungen der $n_i \bmod p$ eine streng monoton fallende Folge bilden. Nach endlich vielen Schritten erhält man auf diese Weise $n_i \equiv 1 \bmod p$ und damit $r_i^2 \equiv a \bmod p$, d.h. $r = r_i$ ist die gesuchte Lösung.

Bekanntermaßen ist die multiplikative Gruppe $G = (\mathbb{Z}/p\mathbb{Z})^*$ zyklisch und hat die Ordnung $(p-1)$. Diese Gruppenordnung kann geschrieben werden in der Form

$$p - 1 = 2^s \cdot (2k+1) \tag{4.24}$$

mit eindeutig bestimmten $s, k \in \mathbb{N}_0$. Die Gruppe G zerfällt also in das direkte Produkt zweier zyklischer Untergruppen G_1 und G_2, wobei G_1 genau die Elemente von G enthält, deren Ordnung eine Zweierpotenz ist, und G_2 diejenigen Elemente, deren Ordnung ungerade ist. Die Ordnung von G_1 ist 2^s, die Ordnung von G_2 ist $(2k+1)$. Für jedes Element $\overline{v} \in G$ gilt damit

$$\overline{v}^{2k+1} \in G_1. \tag{4.25}$$

Ferner gilt: Eine Zahl $v \in \mathbb{Z}$ ist genau dann quadratischer Nichtrest modulo p, wenn die Ordnung von $v \bmod p$ von 2^s geteilt wird.

Wir benötigen nun zunächst ein erzeugendes Element für die Gruppe G_1. Dazu wählen wir eine Zahl $z \in \mathbb{Z}$, die quadratischer Nichtrest modulo p ist, und berechnen

$$c_0 \equiv z^{2k+1} \bmod p. \tag{4.26}$$

Wegen (4.25) liegt $\overline{c_0}$ sicher in G_1. Da z quadratischer Nichtrest modulo p ist, ist auch c_0 quadratischer Nichtrest modulo p, die Ordnung von $c_0 \bmod p$ wird also von 2^s geteilt. Insgesamt folgt, daß $\overline{c_0}$ die Gruppe G_1 erzeugt.

Die Bestimmung des quadratischen Nichtrestes z erfolgt durch Ausprobieren, d.h. man wählt eine Zufallszahl $z \in [\![2, p-1]\!]$ und testet durch Auswertung des Jacobisymbols $(\frac{z}{p})$, ob z quadratischer Nichtrest modulo p ist. Dies wiederholt man so lange, bis man einen quadratischen Nichtrest gefunden hat. Die Auffindung von z ist der einzige probabilistische Schritt im Algorithmus RESSOL, die Erfolgswahrscheinlichkeit ist jedoch hoch (vgl. [24], S. 49). Unter Annahme der Verallgemeinerten Riemannschen Vermutung kann man sogar zeigen, daß der kleinste quadratische Nichtrest von der Größenordnung $O(\log^2 p)$ ist (vgl. [13], S. 26).

Wir setzen nun

$$r_0 \equiv a^{k+1} \bmod p, \quad n_0 \equiv a^{2k+1} \bmod p. \tag{4.27}$$

Bei dieser Wahl der Startwerte ist die Bedingung (4.23) sicher erfüllt. Wegen (4.25) ist außerdem $\overline{n_0} \in G_1 = \langle \overline{c_0} \rangle$. Weiterhin ist n_0 quadratischer Rest modulo p, da a quadratischer Rest modulo p ist, die Ordnung von $n_0 \bmod p$ wird somit von 2^s nicht geteilt. Es gibt also einen Exponenten $e \in \mathbb{N}_0$ mit $2 \mid e$, so daß $n_0 \equiv c_0^e \bmod p$ ist.

An dieser Stelle beginnt der iterative Vorgang der Erzeugung der Folge (r_i, n_i). Dazu setzen wir noch $s_0 = s$ und initialisieren den Schleifenindex i mit dem Wert 0. Die Schleife wird beendet, wenn $n_i \equiv 1 \bmod p$ ist.

Im Laufe der Iteration sollen die folgenden drei Invarianzbedingungen gelten:

(i) Gleichung (4.23) ist erfüllt.

(ii) Es existiert ein Exponent $e \in \mathbb{N}_0$ mit $2 \mid e$, so daß $n_i \equiv c_i^e \bmod p$ gilt, d.h. insbesondere $\overline{n_i} \in \langle \overline{c_i} \rangle$.

(iii) 2^{s_i} ist die Ordnung von $c_i \bmod p$.

Für $i = 0$ sind nach den obigen Betrachtungen alle drei Bedingungen erfüllt.

Da $\overline{n_i} \in \langle \overline{c_i} \rangle$, kann man n_i als Potenz von c_i schreiben, d.h. es gibt eindeutig bestimmte Zahlen $t_i \in \mathbb{N}_0$ und $m_i \in \mathbb{N}$, so daß

$$n_i = c_i^{2^{t_i} \cdot (2m_i - 1)}$$

gilt. Wegen der zweiten Invarianzbedingung ist außerdem $t_i \neq 0$. Die Ordnung von $n_i \bmod p$ ist dann offensichtlich $2^{s_i - t_i}$.

Der Wert t_i kann leicht ermittelt werden, indem man n_i sukzessive quadriert, denn nach $(s_i - t_i)$ Quadrierungen ist das Resultat zum ersten Mal kongruent $1 \bmod p$. Die Bestimmung von m_i wäre wesentlich zeitaufwendiger, sie ist jedoch auch nicht erforderlich.

Nun soll das Folgenglied (r_{i+1}, n_{i+1}) konstruiert werden. Man setzt dazu

$$r_{i+1} \equiv r_i \cdot b_i \bmod p, \quad n_{i+1} \equiv n_i \cdot b_i^2 \bmod p,$$

wobei $b_i \in \mathbb{Z}$ so gewählt wird, daß die Ordnung von $n_{i+1} \bmod p$ kleiner als die Ordnung von $n_i \bmod p$ ist. Offensichtlich gilt für das Paar (r_{i+1}, n_{i+1}) Gleichung (4.23), die erste Invarianzbedingung ist also wieder erfüllt.

Shanks Idee besteht nun darin,

$$b_i \equiv c_i^{2^{t_i-1}} \bmod p$$

zu wählen, so daß wir

$$n_{i+1} \equiv c_i^{2^{t_i}\cdot(2m_i-1)} \cdot c_i^{2^{(t_i)}} \equiv c_i^{2^{t_i+1}\cdot m_i} \bmod p \tag{4.28}$$

erhalten. Die Ordnung von $n_{i+1} \bmod p$ ist somit höchstens $2^{s_i-t_i-1}$, also sicher kleiner als die Ordnung von $n_i \bmod p$.

Wir setzen nun

$$c_{i+1} \equiv b_i^2 \equiv c_i^{2^{t_i}} \bmod p,$$

und

$$s_{i+1} = s_i - t_i.$$

Dann folgt aus der Tatsache, daß 2^{s_i} die Ordnung von $c_i \bmod p$ war, daß $2^{s_{i+1}}$ die Ordnung von $c_{i+1} \bmod p$ ist. Aus Gleichung (4.28) folgt weiterhin unmittelbar, daß $n_{i+1} \equiv c_{i+1}^{2m_i} \bmod p$ ist, d.h. es ist $\overline{n_{i+1}} \in \langle \overline{c_{i+1}} \rangle$, und es ist n_{i+1} eine gerade Potenz von c_{i+1} modulo p.

Damit sind alle Invarianzbedingungen der Iteration erfüllt, und die Rekursion kann mit $i = i + 1$ erneut beginnen.

Aus diesen Überlegungen ergibt sich der folgende Algorithmus zur Berechnung von Quadratwurzeln modulo p:

Algorithmus 4.20 (Quadratwurzeln modulo p)

Eingabe: $a \in \mathbb{Z}$ und $p \in \mathbb{P}, p \neq 2$ mit $(\frac{a}{p}) = 1$
Ausgabe: eine Zahl $r \in [\![1, p-1]\!]$ mit $r^2 \equiv a \bmod p$

Bestimme $k, s \in \mathbb{N}_0$, so daß $p - 1 = 2^s(2k+1)$.		(1)
Setze $r \leftarrow a^{k+1} \bmod p$ und $n \leftarrow a^{2k+1} \bmod p$.		(2)
IF $n \equiv 1 \bmod p$ THEN EXIT.		(3)
Bestimme einen quadratischen Nichtrest z modulo p und setze $c \leftarrow z^{2k+1} \bmod p$.		(4)
	Bestimme u, so daß $n^{2^u} \equiv 1 \bmod p$. Setze $t \leftarrow s - u$ und $s \leftarrow u$.	(5)
	Setze $b \leftarrow c^{2^{t-1}} \bmod p$.	(6)
	Setze $r \leftarrow rb \bmod p$, $c \leftarrow b^2 \bmod p$, und $n \leftarrow nb^2 \bmod p$.	(7)
UNTIL $n \equiv 1 \bmod p$.		(8)

Mit Ausnahme der Bestimmung des quadratischen Nichtrestes in Schritt (3) ist Algorithmus 4.20 ein deterministischer Algorithmus. Man kann leicht verifizieren, daß die Laufzeit des deterministischen Teils polynomiell in $\log p$ ist (vgl. [24], S. 49).

Kapitel 5

Der Algorithmus von Hafner und McCurley

In diesem Kapitel beschreiben wir den Algorithmus zur Klassengruppenberechnung, der von Hafner und McCurley in [18] vorgeschlagen wird. Ziel dieses Kapitels ist es neben der Darstellung des Algorithmus auch, die Aussagen aus [18] über Erfolgswahrscheinlichkeit und Laufzeit des Algorithmus nachzuvollziehen und damit den folgenden Satz zu beweisen.

Satz 5.1 (Hafner, McCurley) *Unter der Annahme der Verallgemeinerten Riemannschen Vermutung existiert ein Las-Vegas-Algorithmus, der für eine Diskriminante $\Delta < -4$, $\Delta \equiv 0, 1 \bmod 4$ die Struktur der Klassengruppe $Cl(\Delta)$ und die Klassenzahl $h(\Delta)$ in erwarteter Laufzeit von $L(|\Delta|)^{\sqrt{2}+o(1)}$ Bit-Operationen berechnet.*

5.1 Der Algorithmus

Wir stellen in diesem Abschnitt den von Hafner und McCurley [18] formulierten Algorithmus zur Klassengruppenberechnung dar. Der Algorithmus baut auf der in Kapitel 4 beschriebenen Idee auf.

Zuvor definieren wir noch für $n, \lambda \in \mathbb{N}$ die Menge

$$W_n(\lambda) = \{x = (x_1, \ldots, x_n) \in \mathbb{Z}^n \ : \ |x_i| \leq \lambda \ (1 \leq i \leq n)\}.$$

Die Parameter m und z des Algorithmus bleiben zunächst unspezifiziert. Sie werden später geeignet gewählt. Die Konstante c_1 entspricht der aus Satz 4.4.

Algorithmus 5.2 (Hafner, McCurley)

Eingabe: Diskriminante $\Delta \in \mathcal{D}^-$
Ausgabe: Klassenzahl $h = h(\Delta)$ und die Struktur der Klassengruppe $Cl(\Delta)$

Berechne eine Abschätzung h^* mit $h^* \le h(\Delta) < 2h^*$.		(1)
Setze $D \leftarrow \lvert\Delta\rvert$ und $N \leftarrow \max\{\lfloor c_1 \log^2 D \rfloor, \lfloor L(D)^z \rfloor\}$. Berechne die Menge $F = \{f_i = (p_i, b_{p_i}) \,:\, (\frac{\Delta}{p_i}) = 1,\ f_i$ Primform in $\mathcal{QF}(\Delta),\ p_i < N\}$. Setze $n \leftarrow \#F$ und $n_0 \leftarrow \max\{i \le n \,:\, p_i < \lfloor c_1 \log^2 D \rfloor\}$.		(2)
	Generiere n dünn besetzte Relationen $w_1, \ldots, w_n$ über F mit Einträgen aus $W_n(n^2 D)$, so daß die Matrix A_0, die aus den Spalten $w_1, \ldots, w_n$ gebildet ist, nicht singulär ist.	(3)
	Berechne $h_0 = \lvert \det A_0 \rvert$.	(4)
	Generiere m neue Relationen $v_1, \ldots, v_m$ über F mit Einträgen aus $W_n(D^2)$.	(5)
	Berechne die Hermite-Normalform H der Matrix $A_1 \in \mathbb{Z}^{n \times (m+n)}$, deren Spalten $w_1, \ldots, w_n, v_1, \ldots, v_m$ sind. Setze $h \leftarrow \operatorname{Det} \Lambda(A_1)$.	(6)
UNTIL $h < 2h^*$.		(7)
Berechne die Smith-Normalform S von H.		(8)
Gib h aus sowie die Diagonaleinträge von S, die größer als 1 sind.		(9)

Die Berechnung der Approximation h^* in Schritt (1) erfolgt wie in Abschnitt 4.3 dargestellt.

Der Parameter N in Schritt (2) ist so gewählt, daß nach den Ausführungen in Abschnitt 4.3 die Menge F eine Formenbasis für die Klassengruppe ist. Bei der Festlegung von n_0 wird vorausgesetzt, daß die Primformen $f_i \in F$ nach der Größe der Primzahlen p_i aufsteigend numeriert sind.

In Schritt (3) müssen dünn besetzte Relationen in der Klassengruppe gefunden werden. Dazu wird Algorithmus 4.11 verwendet. Für die Wahl der Einträge des Exponentenvektors x wird dabei folgendes festgelegt:

- Die Einträge $x_1, \ldots, x_{n_0}$ werden zufällig aus $[\![0, D]\!]$ gewählt.
- Die Einträge $x_{n_0+1}, \ldots, x_n$ werden auf Null gesetzt.
- Bei der Berechnung des k-ten Relationenvektors w_k $(1 \le k \le n)$ wird $2nD$ zu x_k addiert.

Falls D genügend groß ist, sind dann alle Einträge der Relationenvektoren w_k kleiner als $n^2 D$, und die Vektoren w_k sind dünn besetzt.

Durch die Addition von $2nD$ zur k-ten Komponente des Vektors x bei der Erzeugung des k-ten Relationenvektors ist sichergestellt, daß die Matrix A_0 diagonaldominant und daher nicht singulär ist.

Schritt (4) wird mit modularer Gauß-Elimination (Algorithmus 3.35) durchgeführt.

Die Erzeugung der Relationen in Schritt (5) erfolgt wiederum mit Algorithmus 4.11, jedoch werden bei der Wahl der Einträge des Exponentenvektors x folgende Änderungen vorgenommen:

- Die zufälligen Einträge werden aus $[\![-D^2, D^2]\!]$ gewählt.
- Die Einträge $x_{n_0+1}, \ldots, x_n$ werden nicht auf Null gesetzt, sondern ebenfalls zufällig gewählt.
- Die Addition von $2nD$ zu einem der Vektoreinträge entfällt.

Hier werden also weder dünn besetzte Vektoren erzeugt, noch sind die Einträge der Vektoren durch n^2D beschränkt. Auch die Forderung der Diagonaldominanz entfällt.

Für Schritt (6) wird modulare Hermite-Reduktion (Algorithmus 3.22) verwendet, wobei h_0 als Modul dient. Die Zahl $h_0 = |\det A_0|$ ist, da die Spalten der Matrix A_0 in der Matrix A_1 enthalten sind, Vielfaches der Determinante des von A_1 erzeugten Gitters (vgl. Satz 3.12) und kann deshalb als Modul verwendet werden.

Schritt (7) wird mit Algorithmus 3.28 durchgeführt, wobei die Klassenzahl $h = \det H$ als Modul verwendet wird.

Offensichtlich sind sowohl (F, A_0, n, n) als auch $(F, A_1, n, n+m)$ Relationensysteme für die Klassengruppe $Cl(\Delta)$. Insgesamt ergibt sich daher aus unseren Betrachtungen der folgende Satz:

Satz 5.3 *(GRH) Wenn Algorithmus 5.2 bei Eingabe von $\Delta \in \mathcal{D}^-$ terminiert, so werden Klassenzahl $h(\Delta)$ und Struktur der Klassengruppe $Cl(\Delta)$ korrekt ausgegeben.*

5.2 Die Erfolgswahrscheinlichkeit

Wir wollen nun zeigen, daß bei entsprechender Wahl des noch nicht festgelegten Parameters m mit hoher Wahrscheinlichkeit der Algorithmus nach einmaligem Durchlaufen der Schritte (3) bis (7) terminiert. Wir zeigen:

Satz 5.4 *Für genügend großes D gibt es ein $m = n^{1+o(1)}$, so daß Algorithmus 5.2 mit einer Wahrscheinlichkeit von mehr als $(1 - \frac{1}{D})$ nach einmaligem Durchlaufen der Schritte (3) bis (7) terminiert.*

Wir fassen im folgenden die Argumentation aus [18] zum Beweis dieses Satzes zusammen, um die Beweisidee zu verdeutlichen. Für technische Details, insbesondere für die Beweise der verwendeten Lemmata, verweisen wir auf [18].

Im folgenden sei Λ das volle Gitter der Relationen über F. In Schritt (5) des Algorithmus wird eine aufsteigende Folge von Gittern

$$\Lambda_0 \subseteq \Lambda_1 \subseteq \ldots \subseteq \Lambda_m \subseteq \Lambda$$

erzeugt, wobei

$$\Lambda_0 = \langle w_1, \ldots, w_n \rangle$$

und

$$\Lambda_i = \langle w_1, \ldots, w_n, v_1, \ldots, v_i \rangle \qquad (1 \le i \le m).$$

Der Algorithmus terminiert in Schritt (7) genau dann, wenn $\Lambda_m = \Lambda$.

Wir haben bereits gesehen, daß $\dim \Lambda_0 = n$ ist. Λ_0 ist also Teilgitter von Λ mit endlichem Index. $G = \Lambda / \Lambda_0$ ist daher endliche abelsche Gruppe der Ordnung

$$\#G = \frac{\text{Det } \Lambda_0}{\text{Det } \Lambda} = \frac{h_0}{h} \le h_0. \tag{5.1}$$

Wir können Schritt (5) interpretieren als die Erzeugung von Elementen von G, wobei diese Elemente jeweils durch einen ihrer Repräsentanten $w \in \Lambda$ der entsprechenden Nebenklasse $w + \Lambda_0$ gegeben sind.

Wir zeigen nun, daß die Elemente von G in Schritt (5) zufällig und annähernd gleichverteilt gewählt werden und folgern daraus, daß mit hoher Wahrscheinlichkeit $\Lambda_m = \Lambda$ für ein endliches m gilt.

Zunächst wird ein Lemma für eine leicht verallgemeinerte Situation angegeben (Beweis in [18]):

Lemma 5.5 *Es sei Γ ein Gitter im $\mathbb{Z}^n$ mit Dimension n. Es sei F eine Fundamentalmasche von Γ und $D = D(\Gamma)$ eine obere Schranke für den Diameter von F. Es sei $w \in \mathbb{Z}^n$ und*

$$N_\Gamma(t, w) = \#\{v \in \mathbb{Z}^n \; : \; v \in (w + \Gamma) \cap W_n(t)\}.$$

Dann gilt: Falls $t > nD$, folgt

$$N_L(t, w) = \frac{(2t)^n}{Det\,\Gamma}\left(1 + O\left(\frac{nD}{t}\right)\right).$$

Dieses Lemma besagt, daß jede Nebenklasse $w + \Lambda$ ($w \in \mathbb{Z}^n$) ungefähr gleich viele Vertreter in der Box $W_n(t)$ besitzt, falls t groß genug ist. Diese Aussage wird nun im nächsten Lemma angewandt, um zu zeigen, daß Schritt (5) von Algorithmus 4.11 tatsächlich die Wahl zufälliger annähernd gleichverteilter Elemente aus G simuliert. Dazu ist zu zeigen, daß für jedes $w \in \Lambda$ die Wahrscheinlichkeit $P(w)$ dafür, daß w gewählt wird, etwa gleich ist.

Zunächst untersuchen wir $P(w)$ genauer: Eine in Schritt (5) erzeugte Relation hat die Form $e = x - y$ ($e, x, y \in \mathbb{Z}^n$), wobei x zufällig aus $W_n(D^2)$ gewählt wird. Der Vektor y hängt von x ab, d.h. $y = y(x)$, und es gilt $0 \le |y_i| \le \log D$ $(1 \le i \le n)$.

Der Vektor $e = x - y(x)$ kann nur gewählt werden, wenn die reduzierte Form aus der Klasse $\varphi(x)$ über F vollständig zerfällt, d.h. wenn

$$\varphi(x) \; \in \; S = \{[f] \in Cl(\Delta) \; : \; f \text{ zerfällt vollständig über F}\}.$$

Es ist also

$$P(w) = P\Big(x - y(x) \in (w + \Lambda_0) \mid \varphi(x) \in S\Big).$$

Diese bedingte Wahrscheinlichkeit wird mit Hilfe des folgenden Lemmas abgeschätzt, in dessen Beweis auch die Aussage aus Lemma 5.5 eingeht (Beweis in [18]):

Lemma 5.6 *Sei $w \in \Lambda$ fest. Dann ist*

$$P(x - y(x) \in (w + \Lambda_0) \mid \varphi(x) \in S) = \frac{h}{h_0}\left(1 + O\left(\frac{n^3}{D}\right)\right).$$

Lemma 5.6 liefert eine Abschätzung für die Wahrscheinlichkeit, mit der in Schritt (5) eine feste Nebenklasse $w + \Lambda_0 \in G$ gewählt wird. Für genügend großes D ergibt sich tatsächlich eine annähernde Gleichverteilung.

Wir müssen nun noch untersuchen, wie die Wahrscheinlichkeit dafür, daß nach der Wahl von k Elementen aus G die volle Gruppe erzeugt wird, von der Wahrscheinlichkeit abhängt, mit der die einzelnen Elemente aus G gewählt werden. Dazu dient das folgende Lemma (Beweis in [18]):

Lemma 5.7 *Es sei G endliche abelsche Gruppe der Ordnung M mit*

$$G \cong \bigoplus_{p|M} \bigoplus_{i=1}^{\alpha_p} \mathbb{Z}/p^{\beta_i(p)}\mathbb{Z}.$$

Es seien $g_1, \ldots, g_k$ Elemente von G, die zufällig mit Zurücklegen gewählt wurden nach einer Wahrscheinlichkeitsverteilung mit Dichtefunktion P, für die

$$P(g) \leq \alpha/M$$

gilt für jedes $g \in G$.

Es sei $I = \sum_{p|M} \alpha_p$ die Anzahl der primen Invarianten von G. Wir bezeichnen mit $\hat{P}_k$ die Wahrscheinlichkeit dafür, daß $\langle g_1, \ldots, g_k \rangle = G$ ist. Dann gilt:

$$\hat{P}_k \geq 1 - \alpha^k \cdot 2^{I-k}.$$

Weiterhin gilt immer $2^I \leq \#G$, so daß wir

$$\hat{P}_k \geq 1 - \left(\frac{\alpha}{2}\right)^k \cdot 2^I \geq 1 - \left(\frac{\alpha}{2}\right)^k \#G \tag{5.2}$$

erhalten.

Nach Formel (5.1) ist $\#G \leq h_0$. Mit Hilfe der Hadamardschen Ungleichung (3.3) erhalten wir $h_0 = |\det A_0| \leq n^{5n/2}d^n$, denn der betragsgrößte Eintrag in der Matrix A_0 ist beschränkt durch $n^2 D$. Für genügend großes D gilt damit

$$\#G \leq h_0 \leq \exp\left(n^{1+o(1)}\right). \tag{5.3}$$

Nach Lemma 5.6 können wir $\alpha = 1 + O(\frac{n^3}{D})$ setzen. Da $(\frac{n^3}{D})$ gegen Null konvergiert, gilt dann für genügend großes D, daß $\alpha < 2$ ist.

Man kann also für genügend großes D eine Zahl $m = n^{1+o(1)}$ wählen, so daß gilt:

$$m \geq \frac{\log(\#G) + \log d}{\log(2/\alpha)}.$$

Dann folgt (beachte $\alpha < 2$)

$$\left(\frac{\alpha}{2}\right)^m \leq \left(\frac{\alpha}{2}\right)^{\frac{\log(\#G)}{\log(2/\alpha)}} \cdot \left(\frac{\alpha}{2}\right)^{\frac{\log d}{\log(2/\alpha)}} = \frac{1}{\#G} \cdot \frac{1}{D},$$

und wir erhalten mit Formel (5.2) die Abschätzung

$$\hat{P_m} \geq 1 - \left(\frac{\alpha}{2}\right)^m \cdot \#G = 1 - \frac{1}{D},$$

was Satz 5.4 beweist.

5.3 Die Laufzeit

In diesem Abschnitt wird die Laufzeit von Algorithmus 5.2 analysiert. Zunächst wird die Laufzeit der einzelnen Schritte angegeben, anschließend wird der Parameter z so gewählt, daß die Gesamtlaufzeit minimal wird. Alle Aussagen gelten nur für genügend großes D.

In [29] wird gezeigt, daß die Laufzeit der Schritte (1) und (2) maximal $n^{1+o(1)}$ ist. Ferner wird dort gezeigt, daß für Schritt (4) $n^{4+o(1)}$ Bit-Operationen benötigt werden, wenn man modulare Gauß-Elimination und Chinesischen Restsatz benutzt.

Nun wird die Laufzeit der Schritte (3) und (5) abgeschätzt. Diese Schritte werden mit Algorithmus 4.11 durchgeführt.

Die Darstellung in [38] zeigt, daß der Test eines einzigen Exponentenvektors, d.h. der einmalige Durchlauf der Schritte (1) bis (3) von Algorithmus 4.11, erwartete Laufzeit $n^{1+o(1)}$ hat, wenn binäre Exponentiation zur Berechnung des Potenzproduktes verwendet wird. Die Laufzeit der Schritte (5) und (6) von Algorithmus 4.11 ist bedeutend geringer.

Die Wahrscheinlichkeit dafür, daß ein Exponentenvektor eine Relation liefert, ist $L(D)^{-\frac{1}{4z}+o(1)}$ (vgl. ebenfalls [38]). Damit ergibt sich eine erwartete Laufzeit von

$$(n+m)n^{1+o(1)}L(D)^{-\frac{1}{4z}+o(1)}$$

für die Erzeugung aller Relationen.

Im letzten Abschnitt haben wir festgestellt, daß die Wahl von $m = n^{1+o(1)}$ eine genügend große Erfolgswahrscheinlichkeit für den Algorithmus garantiert. Somit ist die erwartete Laufzeit der Schritte (3) und (5)

$$n^{2+o(1)}L(D)^{-\frac{1}{4z}+o(1)}.$$

Für die modulare HNF-Berechnung in Schritt (6) wird der Modul h_0 verwendet. Die Laufzeit ist also nach Satz 3.23

$$O(nm \log h_0 \log \log^3 h_0 + n^2 m \log h_0 \log \log^2 h_0).$$

Setzen wir $m = n^{1+o(1)}$ und $h_0 = \exp(n^{1+o(1)})$ (vgl. (5.3)) ein, so erhalten wir für Schritt (6) mit $n^{4+o(1)}$ dieselbe Laufzeit wie für Schritt (4).

Für die SNF-Berechnung in Schritt (8) wird die Klassenzahl h als Modul verwendet. Unter Benutzung der trivialen Tatsache, daß $h < D$ ist, ergibt sich aus Satz 3.30 eine Laufzeit von

$$O(n^2 \log^3 D \log \log^3 D + n^3 \log^3 D \log \log^2 D),$$

die für genügend großes D durch

$$O(n^3 \log^4 D)$$

abgeschätzt werden kann.

Die Gesamtlaufzeit von Algorithmus 5.2 ist die Summe der hier angegebenen Laufzeiten für die einzelnen Schritte. Diese wird dominiert von der Summe der Laufzeiten der Schritte (3) und (5) sowie der Schritte (4) und (6). Die Gesamtlaufzeit ist also

$$n^{2+o(1)} L(D)^{-1/(4z)+o(1)} + n^{4+o(1)}.$$

Da $n = L(D)^{z+o(1)}$ ist, haben beide Summanden für $z = 1/\sqrt{8}$ denselben Wert. Diese Wahl von z ist also optimal. Damit ist Satz 5.1 bewiesen.

Kapitel 6

Der Algorithmus CLASS-GROUP

6.1 Motivation

In diesem Kapitel beginnen wir mit der Beschreibung des von uns entwickelten Algorithmus CLASSGROUP zur Berechnung imaginär-quadratischer Klassenzahlen und Klassengruppen.

Die Motivation für die Entwicklung eines neuen Algorithmus ist dadurch gegeben, daß der Algorithmus von Hafner und McCurley in der Praxis nicht effizient ist, sondern vielmehr große Probleme bereitet. Betrachtet man nämlich diesen Algorithmus genauer, so erkennt man, daß bei seiner Entwicklung das Ziel der theoretischen Analysierbarkeit gegenüber dem der praktischen Effizienz im Vordergrund stand. Es wurde Wert darauf gelegt, eine Formulierung zu finden, die den Beweis der Komplexitätsaussage in Satz 5.1 ermöglicht. Hafner und McCurley weisen in ihrer Arbeit selbst darauf hin, daß ein in der Praxis effizienter Algorithmus wohl eine andere Form haben müßte.

Im folgenden werden die grundsätzlichen praktischen Probleme des Algorithmus von Hafner und McCurley angegeben und damit die Notwendigkeit der Entwicklung neuer Strategien begründet. Die Effizienzprobleme lassen sich wie folgt zusammenfassen:

- Die Relationenerzeugung ist langsam. Hauptgrund hierfür ist der große Aufwand, der für die Berechnung der Potenzprodukte erforderlich ist.
- Die Determinantenberechnung und die modulare Hermite-Reduktion sind ebenfalls langsam und benötigen viel Speicherplatz. Der Grund hierfür ist die Größe der zu bearbeitenden Matrizen und die Größe der Matrixeinträge.

Die tatsächliche Bedeutung dieser Probleme wird im wesentlichen durch die Größe von $D = |\Delta|$ und die daraus resultierende Anzahl der Elemente in der Formenbasis bestimmt. Wir untersuchen daher zunächst den Einfluß der Größe von D auf den

Algorithmus. Die dabei verwendeten Bezeichnungen sind dieselben wie in Abschnitt 5.1.

Nur für Werte von D, die über 10^{105} liegen, ist $\lfloor L(D)^{\sqrt{2}+o(1)} \rfloor > \lfloor 6 \log^2 D \rfloor$ und damit $n > n_0$. Nur in diesem Fall sind Teile der Matrix A_0, nämlich die Zeilen $(n_0+1), \ldots, n$ dünn besetzt. Man beachte, daß die ersten n_0 Zeilen der Matrix in jedem Fall dicht besetzt sind.

Der für die Praxis zunächst interessante Bereich umfaßt Werte von D zwischen 10^{20} und 10^{40}. Hier ergibt sich also $N = \lfloor 6 \log^2 D \rfloor$, und die Größe n der Formenbasis liegt im Bereich zwischen etwa 1000 und 3000. Ferner hat man in diesen Fällen $n_0 = n$, d.h. die Matrix A_0 ist dicht besetzt.

Wir betrachten nun zur Verdeutlichung der Problemgröße einen Wert von $D = 10^{30}$, der zu einer Formenbasisgröße von etwa 1500 führt.

In diesem Fall müssen für jeden in Schritt (3) bzw. (5) verwendeten Exponentenvektor x, der daraufhin untersucht werden soll, ob er eine Relation liefert, 1500 Potenzen von Formenklassen berechnet und aufmultipliziert werden. Die Größe der Exponenten in diesem Potenzprodukt liegt dabei in Schritt (3) bei 10^{30}, in Schritt (5) sogar bei 10^{60}. Selbst wenn man binäre Exponentiation verwendet, benötigt man pro Potenz in Schritt (3) durchschnittlich etwa 150 Multiplikationen von Formenklassen, in Schritt (5) etwa 300. Pro Exponentenvektor sind also in Schritt (3) etwa 250 000 Multiplikationen von Formenklassen erforderlich, in Schritt (5) etwa 500 000. Dieser Aufwand ist nicht praktikabel.

In Schritt (4) des Algorithmus wird die Determinante der Matrix A_0 berechnet, die für das gewählte Beispiel eine Größe von 1500 Zeilen und Spalten hat und dicht besetzt ist. Die Hadamardsche Ungleichung ergibt bei der von Hafner und McCurley vorgeschlagenen Wahl der Matrixeinträge eine obere Abschätzung von 10^{60000} für die Determinante von A_0. Es sind dann etwa 10^6 Durchläufe der modularen Gauß-Elimination mit Primzahlen der Größe 10^{20000} auf der Matrix A_0 erforderlich, um die Determinante zu berechnen. Rechenzeit- und Speicherplatzaufwand hierfür sind ebenfalls nicht praktikabel.

Für die modulare Hermite-Reduktion in Schritt (6) gelten ähnliche Argumente.

Eine weitere Eigenschaft des Algorithmus von Hafner und McCurley, die nachteilige Auswirkungen auf die Laufzeit hat, ist die Tatsache, daß im Verlauf des Algorithmus in den meisten Fällen fast doppelt so viele Relationen erzeugt werden wie erforderlich sind. Wir gehen hierauf im nächsten Abschnitt genauer ein.

6.2 Das Konzept des Algorithmus CLASSGROUP

Ziel unserer Arbeit ist die Entwicklung eines Klassengruppenalgorithmus, der in der Praxis eine hohe Effizienz aufweist. Dazu müssen insbesondere die im vorigen Abschnitt beschriebenen Effizienzprobleme des Algorithmus von Hafner und McCurley umgangen werden. Daneben sollen aber auch weitere Möglichkeiten zur Beschleunigung ausgenutzt werden.

Ausgangspunkt für die Entwicklung des neuen Algorithmus ist die in Abschnitt 4.3 vorgestellte grundsätzliche Idee, zunächst mit Hilfe eines probabilistischen Verfahrens Relationen über einem Erzeugendensystem der Klassengruppe zu finden und dann mit Hilfe der Berechnung der Determinante des Relationengitters die Klassenzahl zu ermitteln.

Die Grundstruktur des Algorithmus wird im wesentlichen bestimmt von der Abfolge der Schritte zur Generierung von Relationen und zur Reduktion der vorliegenden Relationenmatrizen. Bei den diesbezüglichen Überlegungen ist die Anzahl der insgesamt im Verlauf des Algorithmus zu generierenden Relationen von großer Bedeutung.

In Schritt (3) des Algorithmus von Hafner und McCurley werden n Relationen generiert, in Schritt (5) nochmals $m \geq n$ weitere Relationen. Es ist denkbar, daß bereits wesentlich weniger Relationen ausreichend sind, um das volle Relationengitter zu erzeugen. Andererseits beginnt der Algorithmus wieder neu in Schritt (3), wenn nach Schritt (5) das Relationengitter noch nicht vollständig erzeugt ist. Alle bisher berechneten Relationen gehen in diesem Fall verloren.

Wir versuchen demgegenüber, möglichst genau so viele Relationen zu generieren, wie tatsächlich zur Erzeugung des Relationengitters benötigt werden. Wir haben in vielen praktischen Experimenten festgestellt, daß in der Regel bereits geringfügig mehr als n Relationen ausreichend sind, um das volle Relationengitter zu erzeugen. Im Algorithmus CLASSGROUP werden daher zunächst $(n+5)$ Relationen generiert, dann wird das von diesen Relationen erzeugte Gitter untersucht. Bei Bedarf werden anschließend solange sukzessive neue Relationen generiert, bis das volle Gitter erzeugt ist.

Durch dieses Vorgehen wird zunächst das Ziel erreicht, keine überflüssigen Relationen zu berechnen. Die Anzahl der zu generierenden Relationen wird so gegenüber dem Algorithmus von Hafner und McCurley um etwa die Hälfte herabgesetzt.

Die Tatsache, daß gleich zu Beginn des Algorithmus bereits $(n + 5)$ Relationen berechnet werden, hat weiterhin auch eine positive Auswirkung auf die Phase der Matrixreduktion. Man kann nämlich mit geringfügig mehr als n Relationen einen wesentlich kleineren Modul für die Hermite-Reduktion erhalten. Dieser Zusammenhang wird später deutlich werden.

Aus diesen Überlegungen ergibt sich die im folgenden dargestellte vorläufige Version des Algorithmus CLASSGROUP.

Algorithmus 6.1 (CLASSGROUP (vorläufige Fassung))

Eingabe: Diskriminante $\Delta \in \mathcal{D}^-$
Ausgabe: Klassenzahl $h(\Delta)$ sowie die Struktur der Klassengruppe $Cl(\Delta)$

Berechne eine Abschätzung $h^* \in \mathcal{Q}$ mit $h^* \leq h(\Delta) < 2h^*$.		(1)
Setze $N \leftarrow \lfloor 6 \log^2 \lvert\Delta\rvert \rfloor$. Berechne die Formenbasis $F = \{f_i = (p_i, b_{p_i}) \ : \ f_i$ Primform in $\mathcal{QF}(\Delta),\ p_i < N\}$. Setze $n \leftarrow \#F$.		(2)
Generiere $m = n + 5$ Relationen $w_1, \ldots, w_m$ über F. Es sei $A \in \mathbb{Z}^{n \times m}$ die Matrix mit den Spalten $w_1, \ldots, w_m$.		(3)
Berechne eine Basis H des von den Spalten der Matrix A erzeugten Gitters in Hermite-Normalform.		(4)
	Setze $h \leftarrow \det H$.	(5)
BREAKIF	$h < 2h^*$ (d.h. die Spalten der Matrix H erzeugen das volle Relationengitter über F).	(6)
	Generiere eine neue Relation v über F.	(7)
	Setze $H' \leftarrow H$ und füge v als zusätzliche Spalte an H' an. Berechne eine Basis H des Gitters $\Lambda(H')$ in Hermite-Normalform.	(8)
Berechne die Smith-Normalform S von H und bestimme so die Struktur der Klassengruppe.		(9)

Man beachte, daß in Schritt (4) des Algorithmus vorausgesetzt wird, daß die Relationenmatrix A regulär ist.

Im nächsten Abschnitt erläutern wir, wie man in Schritt (8) zusätzlich zur Struktur auch ein sogenanntes minimales Erzeugendensystem für die Klassengruppe berechnen kann. In den darauffolgenden Abschnitten führen wir einige grundsätzliche Überlegungen zur Realisierung der Schritte (3) und (4) durch. Dabei erläutern wir auch unsere Ansätze zur Umgehung der in Abschnitt 6.1 diskutierten Effizienzprobleme des Algorithmus von Hafner und McCurley. Die Ergebnisse dieser Abschnitte machen noch einige Änderungen und Präzisierungen in der Formulierung des Algorithmus CLASSGROUP erforderlich. Die endgültige Fassung des Algorithmus wird dann in Abschnitt 6.7 angegeben.

6.3 Bestimmung eines minimalen Erzeugendensystems

In Schritt (8) von Algorithmus 6.1 wird die Struktur der Klassengruppe $Cl(\Delta)$ bestimmt, d.h. es werden ganze Zahlen $m_1, \ldots, m_t > 1$ $(t \leq n)$ bestimmt mit

$$m_i \mid m_{i+1} \ (1 \leq i \leq t-1),$$
$$h(\Delta) = \prod_{i=1}^{t} m_i,$$
$$Cl(\Delta) \cong \mathbb{Z}/m_1\mathbb{Z} \bigoplus \ldots \bigoplus \mathbb{Z}/m_t\mathbb{Z}.$$

Existenz und Eindeutigkeit der m_i wurden bereits in Abschnitt 2.1 begründet.

Wir wissen dann, daß die Klassengruppe direktes Produkt von t vielen zyklischen Untergruppen der Ordnungen $m_1, \ldots, m_t$ ist. Über diese Daten hinaus, die im Algorithmus von Hafner und McCurley ebenfalls berechnet werden, interessieren wir uns nun auch für die Bestimmung von Erzeugern für jede dieser zyklischen Untergruppen.

Die dazu notwendigen theoretischen Überlegungen werden hier in allgemeiner Form durchgeführt. Wir definieren zunächst den Begriff des minimalen Erzeugendensystems für beliebige endliche abelsche Gruppen:

Definition 6.2 *Es sei G eine endliche abelsche Gruppe und $(m_1, \ldots, m_t)$ die Struktur von G. Ein Tupel $(g_1, \ldots, g_t)$ von Elementen aus G mit der Eigenschaft, daß*

$$G = \langle g_1 \rangle \times \ldots \times \langle g_t \rangle,$$
$$\#\langle g_i \rangle = m_i \quad (1 \le i \le t),$$

heißt **minimales Erzeugendensystem** *von G.*

Wir untersuchen nun, wie man ein minimales Erzeugendensystem für eine Gruppe $G = \mathbb{Z}^n/\Gamma$ berechnen kann, wobei Γ ein n-dimensionales Gitter im $\mathbb{Z}^n$ ist. Dazu benötigen wir zunächst den folgenden Hilfssatz im Zusammenhang mit der Smith-Normalform des Gitters Γ.

Satz 6.3 *Es sei Γ ein n-dimensionales Gitter im $\mathbb{Z}^n$ und $S \in \mathbb{Z}^{n \times n}$ die (eindeutig bestimmte) Smith-Normalform von Γ. Dann gibt es eine Matrix $X \in \mathcal{SL}(n)$, so daß die Spalten der Matrix XS eine Basis des Gitters Γ sind.*

Beweis: Es sei $C \in \mathbb{Z}^{n \times n}$ eine Basis von Γ. Dann existieren nach Satz 3.25 Matrizen $U, V \in \mathcal{SL}(n)$, so daß $S = UCV$. Dabei ist offensichtlich CV eine Basis von Γ, und es gilt $XS = CV$. Die Matrix $X = U^{-1}$ hat also die geforderte Eigenschaft. ■

Der folgende Satz stellt nun den Zusammenhang zwischen der Berechnung der Smith-Normalform von Γ und der Bestimmung eines minimalen Erzeugendensystems für G her. Für einen Vektor $v \in \mathbb{Z}^n$ und ein Gitter Γ im $\mathbb{Z}^n$ bezeichnet dabei $[v]$ die Restklasse von v in der Gruppe $G = \mathbb{Z}^n/\Gamma$.

Satz 6.4 *Es seien Γ, S und X wie in Satz 6.3. Es seien $s_1, \ldots, s_n \in \mathbb{Z}_{>0}$ die Diagonaleinträge von S und $k \in [\![1, n]\!]$ der kleinste Index mit $s_k > 1$. Ferner seien $X_1, \ldots, X_n \in \mathbb{Z}^n$ die Spalten von X. Dann ist $([X_k], \ldots, [X_n])$ ein minimales Erzeugendensystem der Gruppe $G = \mathbb{Z}^n/\Gamma$.*

Beweis: Da $X \in \mathcal{SL}(n)$, ist $(X_1, \ldots, X_n)$ ein Erzeugendensystem von $\mathbb{Z}^n$, also ist $([X_1], \ldots, [X_n])$ ein Erzeugendensystem von G. Nach Satz 6.3 liegen die Vektoren $s_1X_1, \ldots, s_nX_n$ in Γ. Für $1 \le j < k$ ist aber $s_j = 1$ und deshalb $X_j \in \Gamma$. Somit ist bereits $([X_k], \ldots, [X_n])$ ein Erzeugendensystem von G. Nun ist nach Satz 3.26 $(s_k, \ldots, s_n)$ die Struktur von G. Für $k \leq j \leq n$ ist ferner s_i ein Vielfaches der Ordnung von $[X_i]$ in G. Da aber $\#G = s_k \cdot \ldots \cdot s_n$ ist, folgt aus der Tatsache, daß $([X_k], \ldots, [X_n])$ ein Erzeugendensystem von G ist, daß s_j die exakte Ordnung von $[X_j]$ in G ist für $k \leq j \leq n$. Also ist $([X_k], \ldots, [X_n])$ ein minimales Erzeugendensystem von G. ■

Wir wenden nun diesen Satz an, um die zu Beginn dieses Abschnitts gestellte Frage nach der Berechnung eines minimalen Erzeugendensystems der Klassengruppe $Cl(\Delta)$ zu beantworten. In Algorithmus 6.1 liegt zu Beginn von Schritt (8) eine Basis H des Gitters Γ der Relationen in $Cl(\Delta)$ über der Formenbasis $F = \{f_1, \ldots, f_n\}$ vor. Mit Hilfe von Satz 6.4 können wir ein minimales Erzeugendensystem $([X_k], \ldots, [X_n])$ von $\mathbb{Z}^n/\Gamma$ angeben. Da $Cl(\Delta) \sim \mathbb{Z}^n/\Gamma$ ist, können wir daraus leicht ein minimales Erzeugendensystem von $Cl(\Delta)$ konstruieren. Dazu muß man lediglich die Vektoren $X_k, \ldots, X_n$ mittels des in Abschnitt 4.3 definierten Homomorphismus φ in die Klassengruppe abbilden. Wir setzen also

$$[g_j] = \prod_{i=1}^{n} [f_i]^{x_{ij}} \qquad (k \leq j \leq n)$$

und erhalten so ein minimales Erzeugendensystem $([g_k], \ldots, [g_n])$ von $Cl(\Delta)$.

Die Berechnung eines minimalen Erzeugendensystems kann also ohne großen zusätzlichen Aufwand parallel zur Berechnung der Struktur der Klassengruppe erfolgen. Man muß dazu lediglich während der Smith-Reduktion das Inverse der linksseitigen Transformationsmatrix ermitteln. Einen entsprechenden Algorithmus geben wir in Kapitel 11 an. Durch diese Ergänzung wird die Ausgabe des Algorithmus CLASSGROUP um eine wichtige Aussage über die Klassengruppe erweitert.

6.4 Die Generierung der Relationen

Wir haben zur Umgehung der in Abschnitt 6.1 genannten Effizienzprobleme für die Generierung der Relationen eine Strategie entwickelt, bei der in den Exponentenvektoren nur etwa 20 Einträge von Null verschieden sind. Einzelheiten dieser Strategie werden in Kapitel 8 beschrieben.

Insgesamt sind bei dieser Vorgehensweise für den Test eines einzelnen Exponentenvektors nur noch jeweils etwa 20 Multiplikationen von Formenklassen notwendig. Der Rechenaufwand für die Berechnung der Potenzprodukte wird damit gegenüber dem Algorithmus von Hafner und McCurley um mehr als vier Zehnerpotenzen herabgesetzt.

Unsere Experimente haben ergeben, daß dieses Vorgehen sehr effizient ist. Insbesondere hat die dünne Besetzung der Exponentenvektoren keinen negativen Einfluß auf die Erfolgswahrscheinlichkeit bei der Generierung der Relationen.

Ein weiterer Effekt unserer Vorgehensweise ist es, daß eine extrem dünn besetzte Relationenmatrix entsteht, in der zudem ein großer Teil der Einträge den Wert ± 1 hat. Im einzelnen hat die Relationenmatrix die folgenden Eigenschaften:

- Nur die ersten 20 bis 30 Zeilen der Matrix sind dicht besetzt, alle weiteren Zeilen sind dünn besetzt.
- Die Anzahl der von Null verschiedenen Einträge pro Zeile nimmt zudem nach unten hin stark ab. In der unteren Hälfte der Matrix haben die meisten Zeilen sogar nur einen oder zwei von Null verschiedene Einträge.
- Die Größe der Matrixeinträge liegt in den ersten 20 bis 30 Zeilen der Matrix in fast allen Fällen betragsmäßig unterhalb von 50, in den übrigen Zeilen der Matrix noch deutlich tiefer.
- In der unteren Hälfte der Matrix haben zudem die meisten Einträge den Wert ± 1, fast alle übrigen den Wert ± 2.

Diese spezielle Struktur der Relationenmatrix kann in der Phase der Matrixreduktion ausgenutzt werden.

In bezug auf den Gesamtablauf des Algorithmus CLASSGROUP ist allerdings zu beachten, daß die mit unserer Strategie generierte Relationenmatrix nicht mit Sicherheit regulär ist. Bei der in Schritt (4) folgenden Matrixreduktion muß also auch die Dimension des von der Relationenmatrix erzeugten Gitters untersucht werden. Bei Bedarf muß das Gitter dann durch die Berechnung geeigneter zusätzlicher Relationen volldimensional gemacht werden.

In einigen speziellen Fällen ist man bereits am Ende von Schritt (3) sicher, daß die Relationenmatrix singulär ist. In diesen Fällen werden sofort weitere Relationen generiert, so daß dann bereits hier geringfügig mehr als $(n+5)$ Relationen vorhanden sind. Ein solcher Fall liegt z.B. dann vor, wenn in der Relationenmatrix eine ganze Zeile nur Nullen enthält.

Im Zusammenhang mit der Generierung der Relationen ist auch die Größe der Formenbasis von Bedeutung. Durch eine Erweiterung der Formenbasis kann die Erfolgswahrscheinlichkeit bei der Generierung der Relationen erhöht werden. Andererseits vergrößert sich dadurch aber die Relationenmatrix, was bei der Matrixreduktion in Schritt (4) des Algorithmus Probleme verursachen kann.

Wir verzichten generell auf die Vergrößerung der Formenbasis durch die Hinzunahme von Primformen mit Primzahlen oberhalb der durch Satz 4.4 vorgegebenen Schranke. Wir vergrößern die Formenbasis allerdings geringfügig durch einige speziell gewählte Formen, deren besondere Eigenschaften zu einer deutlichen Erhöhung der Erfolgswahrscheinlichkeit bei der Generierung der Relationen führen. Die entsprechenden Formen werden in Abschnitt 7.2 im Zusammenhang mit der Beschreibung unseres Verfahrens zur Konstruktion der Formenbasis angegeben.

Alle in diesem Abschnitt angegebenen Eigenschaften unseres Verfahrens zur Generierung der Relationen werden in Kapitel 8 ausführlich erläutert. Dort werden auch noch weitere, hier nicht erwähnte Maßnahmen zur Beschleunigung des Verfahrens beschrieben.

6.5 Standardverfahren für die Matrixreduktion

Grundsätzliches Ziel der Matrixreduktion in Schritt (4) von Algorithmus 6.1 ist es, eine Basis H des von den zuvor generierten Relationen erzeugten Gitters zu bestimmen. Wir gehen zunächst, wie bei der Formulierung des Algorithmus in Abschnitt 6.2, davon aus, daß die Relationenmatrix A nicht singulär ist. Die zu konstruierende Basis H von $\Lambda(A)$ soll in Dreiecksform vorliegen, damit die Determinante des Gitters leicht bestimmt werden kann. Hat H sogar Hermite-Normalform, so weiß man, daß die einzelnen Einträge nicht zu groß sind.

Die Konstruktion einer solchen Gitterbasis H in HNF ist mit dem in Kapitel 3 beschriebenen Algorithmus zur modularen Hermite-Reduktion (Algorithmus 3.22) möglich. Voraussetzung für die Anwendung dieses Algorithmus ist die Kenntnis eines Vielfachen d der Gitterdeterminante D.

Die Berechnung eines solchen Vielfachen d von D kann mit Hilfe der ebenfalls in Kapitel 3 beschriebenen modularen Gauß-Elimination (Algorithmus 3.31) geschehen, die auf einer regulären quadratischen Teilmatrix $A' \in \mathbb{Z}^{n \times n}$ von A durchgeführt wird. Die Spalten dieser Teilmatrix A' erzeugen ein Teilgitter von $\Lambda(A)$, und daher ist die Determinante der Matrix A' ein Vielfaches der Determinante des Gitters $\Lambda(A)$ (vgl. Satz 3.12). Da wir, wie bereits erwähnt, die Relationenmatrix A zunächst als regulär voraussetzen, existiert eine reguläre quadratische Teilmatrix $A' \in \mathbb{Z}^{n \times n}$.

Man erhält so den folgenden Algorithmus zur Matrixreduktion:

Algorithmus 6.5 (Matrixreduktion mit Standardverfahren)

Eingabe: reguläre Matrix $A \in \mathbb{Z}^{n \times m}$
Ausgabe: Basis H des Gitters $\Lambda(A)$ in Hermite-Normalform

Wähle eine reguläre Teilmatrix $A' \in \mathbb{Z}^{n \times n}$ von A und berechne mit Hilfe modularer Gauß-Elimination $d = \det A'$.	(1)
Berechne mit Hilfe modularer Hermite-Reduktion eine Basis H des Gitters $\Lambda(A)$ in Hermite-Normalform. Verwende dabei d als Modul.	(2)

Die Matrixreduktion läßt sich also mit den in Kapitel 3 beschriebenen Standardverfahren durchführen. Es ist lediglich noch ein Algorithmus anzugeben, mit dem die Auswahl der regulären Teilmatrix durchgeführt werden kann.

Bei der Durchführung von Algorithmus 6.5 für eine beliebige große Matrix A treten jedoch in der Praxis einige schwerwiegende Effizienzprobleme auf. Diese Probleme wurden teilweise bereits in Abschnitt 6.1 angedeutet.

Nun ist aber die Matrix A in unserer Anwendung extrem dünn besetzt und hat zudem nur sehr kleine Einträge. Man erwartet, daß die Reduktion einer solchen Matrix sehr viel effizienter möglich ist als die einer beliebigen Matrix. Es zeigt sich aber, daß bei der Realisierung der Reduktion dünn besetzter Matrizen mit den oben angegebenen Standardverfahren dieselben Probleme auftreten wie bei der Reduktion einer beliebigen Matrix. Wir geben im folgenden diese Probleme an und erläutern, warum sie auch bei der Reduktion dünn besetzter Matrizen auftreten.

Das erste Effizienzproblem besteht darin, daß sowohl modulare Gauß-Elimination als auch modulare Hermite-Reduktion bei der Bearbeitung einer großen Matrix viel Speicherplatz benötigen und eine hohe Laufzeit haben. Die dünn besetzte Struktur der in unserer Anwendung vorliegenden Matrix wird nun weder von der Gauß-Elimination noch von der Hermite-Reduktion in irgendeiner Weise ausgenutzt; sie geht vielmehr bei der Verwendung dieser Algorithmen bereits nach kurzer Zeit verloren. Eine dünne Besetzung der Matrix führt hier also weder zu einer Verbesserung der Laufzeit, noch ermöglicht sie eine Einsparung von Speicherplatz durch die Verwendung besonderer Datenstrukturen, in denen nur die wenigen Matrixelemente, die von Null verschieden sind, abgespeichert werden müßten.

Das zweite Effizienzproblem besteht in der hohen Anzahl der Durchläufe der modularen Gauß-Elimination. Diese ist notwendig, da man mit Hilfe der Hadamardschen Ungleichung (3.3) eine sehr große obere Abschätzung für die Determinante von A' erhält. Nun ist die mit Hilfe der Hadamardschen Ungleichung berechnete obere Abschätzung $\tilde{d}$ für $\det A'$ bei einer dünn besetzten Matrix, deren von Null verschiedene Einträge zudem sehr klein sind, tatsächlich wesentlich kleiner als bei einer beliebigen Matrix. Man erhält jedoch gemäß Gleichung (3.4) in jedem Fall $\tilde{d} \geq (\sqrt{n})^n$, d.h. die Anzahl der Durchläufe der modularen Gauß-Elimination wird auch von der Größe der Matrix entscheidend beeinflußt.

Das dritte Effizienzproblem ergibt sich aus der Beobachtung, daß die Zahl $D = \det A'$ in den meisten Fällen ein sehr großes Vielfaches von $d = \mathrm{Det}\,\Lambda(A)$ ist. Für die modulare Hermite-Reduktion muß daher ein sehr großer Modul verwendet werden, was sich - zusätzlich zur Matrixgröße - nochmals nachteilig auf Laufzeit und Speicherplatzbedarf dieses Algorithmus auswirkt. Bei näherer Betrachtung zeigt sich, daß dieses Problem von der Struktur der Matrix völlig unabhängig ist. Es liegt vielmehr in der Tatsache begründet, daß wir bei der Auswahl einer regulären Teilmatrix A' von A mehr oder weniger zufällig ein Teilgitter von $\Gamma = \Lambda(A)$ festlegen, dessen Determinante wir zunächst berechnen. Ein solches durch die Auswahl von n beliebigen Vektoren des Gitters Γ bestimmtes Teilgitter hat in der Regel eine wesentlich größere Determinante als das Gitter Γ selbst.

Zusammenfassend läßt sich festhalten, daß auch die Reduktion einer dünn besetzten Matrix mit dem in Algorithmus 6.5 beschriebenen Vorgehen nicht effizient realisierbar ist. Die einzelnen Probleme werden zwar eventuell geringfügig abgemildert, aber sicherlich nicht befriedigend gelöst. Das Problem der mangelnden Effizienz liegt hauptsächlich in der Tatsache begründet, daß die Matrix A sehr groß ist. Dies führt insgesamt zu großem Rechenzeit- und Speicherplatzaufwand bei modularer Gauß-Elimination und modularer Hermite-Reduktion.

6.6 Neue Ansätze zur Matrixreduktion

Zur Umgehung der im letzten Abschnitt beschriebenen Effizienzprobleme bei der Matrixreduktion haben wir ein neues Verfahren entwickelt, das die besonderen Eigenschaften der in unserer Anwendung vorliegenden Relationenmatrix ausnutzt. Für dieses Verfahren wurden einige neue Algorithmen formuliert, die in den Kapiteln 9 und 10 ausführlich beschrieben werden. Daneben spielen auch die in Algorithmus 6.5 enthaltenen Standardverfahren eine Rolle. Wir erläutern in diesem Abschnitt die grundsätzlichen Ideen des Verfahrens.

Eine dieser grundlegenden Ideen unseres Verfahrens zur Matrixreduktion ist es, die Relationenmatrix zunächst unter Ausnutzung ihrer dünn besetzten Struktur stark zu verkleinern. Dieser Ansatz beruht auf der Beobachtung, daß unter bestimmten Bedingungen auch das von den Spalten einer kleineren Matrix erzeugte Gitter für die Berechnung der Klassengruppe verwendet werden kann.

Ausgangspunkt für diesen Teil der Matrixreduktion ist das Relationensystem $\mathcal{R} = (F, A, n, m)$ mit der in Schritt (2) von Algorithmus 6.1 berechneten Formenbasis F und der in Schritt (3) berechneten Relationenmatrix A sowie $m \geq n + 5$. Ziel ist es, aus diesem Relationensystem ein neues Relationensystem $\mathcal{R}_1 = (F_1, B, n_1, m_1)$ zu konstruieren, wobei n_1 möglichst klein sein soll. Die Verkleinerung der Relationenmatrix muß also so vorgenommen werden, daß die Spalten der entstehenden Matrix $B \in \mathbb{Z}^{n_1 \times m_1}$ Relationen über einer ebenfalls neu zu konstruierenden Formenbasis F_1 sind.

Das Verfahren zur Konstruktion des Relationensystems $\mathcal{R}_1$ wird durch den folgenden Satz motiviert:

Satz 6.6 *Es sei $\Delta \in \mathcal{D}^-$. Es sei $\mathcal{R} = (F, A, n, m)$ ein Relationensystem für die Klassengruppe $Cl(\Delta)$. Es seien $i \in [\![1, n]\!]$ und $j \in [\![1, m]\!]$, so daß*

$$\begin{aligned} a_{ij} &= \pm 1, \\ a_{ik} &= 0 \qquad (1 \leq k \leq m,\ k \neq j). \end{aligned} \tag{6.1}$$

Es sei ferner A' die Matrix, die aus A durch Streichen der i-ten Zeile und der j-ten Spalte entsteht und $F' = F \setminus \{f_i\}$. Dann gilt:

(i) $\mathcal{R}' = (F', A', n-1, m-1)$ ist ein Relationensystem für $Cl(\Delta)$.

(ii) Die Matrix A' ist genau dann regulär, wenn A regulär ist, und in diesem Fall ist $Det\,\Lambda(A') = Det\,\Lambda(A)$.

Beweis: (i) Da $a_{ij} = \pm 1$ ist, folgt

$$[f_i] = \prod_{k=1}^{i-1} [f_k]^{a_{ij} \cdot a_{kj}} \cdot \prod_{k=i+1}^{n} [f_k]^{a_{ij} \cdot a_{kj}},$$

es ist also $[f_i]$ als Potenzprodukt der zu den Formen aus F' gehörenden Äquivalenzklassen darstellbar. Somit ist F' eine Formenbasis für die Klassengruppe. Da außerdem $a_{ik} = 0$ ist für $1 \leq k \leq n,\ k \neq j$, sind alle Spalten der Matrix A' Relationen bezüglich der Menge F'. Damit ist gezeigt, daß $\mathcal{R}'$ ein Relationensystem ist.

(ii) Es sei $m = \dim \Lambda(A)$. Man kann leicht verifizieren, daß eine Basis $C \in \mathbb{Z}^{n \times m}$ des Gitters $\Lambda(A)$ existiert, deren erste Spalte die Spalte A_j ist und für die

$$c_{k1} = 0 \qquad (2 \leq k \leq m)$$

ist. Die Matrix $C' \in \mathbb{Z}^{(n-1)\times(m-1)}$, die aus C durch Streichen der ersten Spalte und der i-ten Zeile entsteht, ist dann eine Basis von $\Lambda(A')$. Da $c_{i1} = \pm 1$ ist, gilt $|\det C| = |\det C'|$, woraus (ii) folgt. ■

Mit Hilfe mehrfacher Anwendung dieses Satzes wird die Relationenmatrix durch das Streichen geeigneter Zeilen und Spalten sukzessive verkleinert. Zur Herstellung der Eliminationsbedingung (6.1) ist dabei teilweise die Durchführung bestimmter Spaltenoperationen erforderlich. Hier wird die dünn besetzte Struktur der Matrix ausgenutzt, indem nur solche Spaltenoperationen durchgeführt werden, die lediglich geringe Auswirkungen auf die Besetzung der Matrix und die Größe der Matrixeinträge haben. So bleibt die dünne Besetzung der Matrix lange erhalten, und wir erreichen eine sehr weitgehende Verkleinerung der Relationenmatrix. In den meisten Fällen beträgt die Größe n_1 der Matrix B nur noch etwa ein Zehntel der ursprünglichen Matrixgröße n. Zudem sind viele Spalten der Matrix B ähnlich dünn besetzt wie die Spalten der Ausgangsmatrix A.

Die entsprechenden Algorithmen zur Verkleinerung der Relationenmatrix werden in Kapitel 9 ausführlich beschrieben.

Man beachte, daß es, bedingt durch die während der Verkleinerung der Relationenmatrix durchgeführten Spaltenoperationen, durchaus möglich ist, daß die Matrix B eine oder mehrere Spalten enthält, die nur aus Nullen bestehen. Man beachte ferner, daß die durch mehrfache Anwendung von Satz 6.6 konstruierte neue Formenbasis F' eine Teilmenge der ursprünglichen Formenbasis F ist.

Nach Satz 4.10 reicht es nun für die Berechnung der Klassenzahl $h(\Delta)$ und der Klassengruppe $Cl(\Delta)$ aus, das volle Relationengitter über der verkleinerten Formenbasis F_1 zu bestimmen und zu untersuchen. Die Untersuchung des Relationengitters Γ über der Formenbasis F wird also ersetzt durch die Untersuchung des Relationengitters Γ_1 über der Formenbasis F_1, das eine wesentlich kleinere Dimension hat.

Im weiteren Verlauf der Matrixreduktion wird also die Hermite-Normalform des von den Spalten der Matrix B erzeugten Gitters berechnet. Dazu wird modulare Gauß-Elimination und modulare Hermite-Reduktion verwendet. Diese rechenzeit- und speicherplatz-intensiven Verfahren müssen nun nicht mehr auf der ursprünglichen großen Relationenmatrix A, sondern nur auf der sehr viel kleineren Matrix B durchgeführt werden. Die Verkleinerung der Relationenmatrix führt somit zur Lösung der ersten beiden in Abschnitt 6.5 genannten Effizienzprobleme.

Zur Lösung des dritten Effizienzproblems ist es erforderlich, ein möglichst kleines Vielfaches d von $D = \operatorname{Det} \Lambda(B)$ als Modul für die Hermite-Reduktion zu bestimmen. Dazu nutzen wir die Tatsache aus, daß die Matrix B mehr Spalten als Zeilen aufweist. Diese Tatsache ermöglicht es, wie in Kapitel 10 bewiesen wird, zwei verschiedene reguläre quadratische n_1-reihige Teilmatrizen B' und B'' von B auszuwählen, falls B selbst regulär ist und mindestens $(n_1 + 1)$ von Null verschiedene Spalten enthält. Wir bestimmen dann mit modularer Gauß-Elimination die Determinanten d' und

d'' dieser beiden Teilmatrizen und setzen $d = \gcd(d', d'')$. Nach Satz 3.12 erzeugen die beiden Teilmatrizen B' und B'' jeweils ein Teilgitter von $\Lambda(B)$, und d' und d'' sind daher Vielfache der Determinante D des Gitters $\Lambda(B)$. Damit ist auch $d = \gcd(d', d'')$ ein Vielfaches von D. Es zeigt sich, daß man auf diese Weise die Größe von d stark herabsetzen kann; in den meisten Fällen erhalten wir $d < 10 \cdot D$.

Wir beschreiben die Algorithmen zur Auswahl der beiden Teilmatrizen und zur Bestimmung von d' und d'' in Kapitel 10. Entscheidend für die Effizienz unseres Verfahrens ist die Tatsache, daß die Berechnung von $d' = \det B'$ und $d'' = \det B''$ simultan erfolgen kann und nicht wesentlich aufwendiger ist als die gewöhnliche modulare Gauß-Elimination auf einer der beiden Teilmatrizen.

Insgesamt ergibt sich aus den Überlegungen in diesem Abschnitt der im folgenden dargestellte Algorithmus zur Realisierung der Matrixreduktion. Man beachte, daß bei der Formulierung dieses Algorithmus die Relationenmatrix A - wie schon in Algorithmus 6.1 - als regulär vorausgesetzt wird. Ferner wird angenommen, daß die in Schritt (1) konstruierte Matrix B mindestens $(n_1 + 1)$ von Null verschiedene Spalten enthält.

Algorithmus 6.7 (Matrixreduktion)

Eingabe: ein Relationensystem $\mathcal{R} = (F, A, n, m)$ mit $m > n$, wobei $A \in \mathbb{Z}^{n \times m}$ regulär ist

Ausgabe: ein Relationensystem $\mathcal{R}_1 = (F_1, H, n_1, n_1)$ mit $n_1 < n$, wobei H in Hermite-Normalform vorliegt

Verkleinere die Relationenmatrix A unter Ausnutzung ihrer dünn besetzten Struktur und konstruiere so aus dem Relationensystem $\mathcal{R} = (F, A, n, m)$ das Relationensystem $\mathcal{R}_1 = (F_1, B, n_1, m_1)$ mit $n_1 < n$, $m_1 = n_1 + (m - n)$, $\operatorname{Det} \Lambda(A) = \operatorname{Det} \Lambda(B)$ und $F_1 \subset F$.	(1)
Wähle zwei verschiedene reguläre Teilmatrizen $B', B'' \in \mathbb{Z}^{n_1 \times n_1}$ von B und berechne mit Hilfe einer modifizierten Form der modularen Gauß-Elimination simultan die Determinanten d' und d'' von B' und B''. Setze $d \leftarrow \gcd(d', d'')$.	(2)
Berechne mit Hilfe modularer Hermite-Reduktion eine Basis H des Gitters $\Lambda(B)$ in Hermite-Normalform. Verwende dabei d als Modul.	(3)

6.7 Die endgültige Fassung des Algorithmus CLASS-GROUP

Bei unseren bisherigen Betrachtungen haben wir vorausgesetzt, daß die in Schritt (3) von Algorithmus 6.1 generierte Relationenmatrix regulär ist. In Abschnitt 6.4 wurde bereits erwähnt, daß dies nicht notwendigerweise immer der Fall ist. Wir untersuchen nun noch, welche Modifikationen an den Algorithmen 6.1 und 6.7 zur Berücksichtigung dieser Tatsache vorzunehmen sind.

Schritt (1) des im vorigen Abschnitt angegebenen Algorithmus 6.7 zur Matrixreduktion kann auch dann unverändert durchgeführt werden, wenn die Relationenmatrix A singulär ist. Aus Satz 6.6 folgt allerdings, daß dann auch die Matrix B singulär ist. In Schritt (2) von Algorithmus 6.7 können daher keine regulären Teilmatrizen von B ausgewählt werden. Eine eventuelle Singularität von B wird also spätestens bei dem Versuch, eine reguläre n_1-reihige Teilmatrix von B auszuwählen, bemerkt. Um nun die Matrixreduktion sinnvoll fortsetzen zu können, ist es notwendig, schon an dieser Stelle eine oder mehrere zusätzliche Relationen zu generieren. Die Matrix B muß dabei so lange durch neue Relationen erweitert werden, bis sie regulär ist.

Schließlich ist noch die Möglichkeit zu berücksichtigen, daß die Matrix B zwar regulär ist, aber nur n_1 von Null verschiedene Spalten enthält. Auch in diesem Fall muß B durch zusätzliche Relationen erweitert werden, da sonst, wie bereits erwähnt, die Auswahl zweier regulärer n_1-reihiger Teilmatrizen von B nicht möglich ist.

Wir formulieren nun die endgültige Version des Algorithmus CLASSGROUP. Dabei werden die in den einzelnen Abschnitten dieses Kapitels erzielten Resultate eingebracht, die wir vorab nochmals kurz zusammenfassen.

- Über die in Algorithmus 6.1 angegebenen Ausgabedaten hinaus wird ein minimales Erzeugendensystem für die Klassengruppe berechnet (vgl. Abschnitt 6.3).
- Die Formenbasis wird um einige geeignete Formen erweitert, um die Erfolgswahrscheinlichkeit bei der Generierung der Relationen zu erhöhen (vgl. Abschnitt 6.4).
- Es wird eine dünn besetzte Relationenmatrix generiert, die eventuell geringfügig mehr als $n + 5$ Spalten hat und nicht notwendigerweise regulär ist (vgl. Abschnitt 6.4).
- Zur Umgehung der Effizienzprobleme bei der Matrixreduktion wurde eine neue Vorgehensweise entwickelt (vgl. Abschnitt 6.6).
- Unter bestimmten Bedingungen, insbesondere dann, wenn die in Schritt (3) berechnete Relationenmatrix singulär ist, müssen schon während der Matrixreduktion zusätzliche Relationen generiert werden (vgl. die Ausführungen zu Beginn dieses Abschnittes).

Man beachte, daß die einzelnen Schritte des Algorithmus 6.7 nunmehr in den Algorithmus CLASSGROUP integriert werden.

Man beachte ferner, daß in den Schritten (7) und (11) die neuen Relationen über der verkleinerten Formenbasis F_1 und nicht über der ursprünglichen Formenbasis F generiert werden müssen.

Algorithmus 6.8 (CLASSGROUP (endgültige Fassung))

Eingabe: Diskriminante $\Delta \in \mathcal{D}^-$

Ausgabe: Klassenzahl $h(\Delta)$ sowie die Struktur und ein minimales Erzeugendensystem der Klassengruppe $Cl(\Delta)$

Berechne eine Abschätzung $h^* \in \mathcal{Q}$ mit $h^* \le h(\Delta) < 2h^*$.	(1)
Setze $N \leftarrow \lfloor 6 \log^2 \lvert\Delta\rvert \rfloor$. Berechne eine Formenbasis F, die die Menge $\{f_i = (p_i, b_{p_i}) \;:\; f_i \text{ Primform in } \mathcal{QF}(\Delta),\ p_i < N\}$ umfaßt. Setze $n \leftarrow \#F$.	(2)
Generiere $m \ge n+5$ dünn besetzte Relationen $w_1, \ldots, w_m$ über F. Es sei $A \in \mathbb{Z}^{n \times m}$ die Matrix mit den Spalten $w_1, \ldots, w_m$.	(3)
Verkleinere die Relationenmatrix A unter Ausnutzung ihrer dünn besetzten Struktur und konstruiere so aus dem Relationensystem $\mathcal{R} = (F, A, n, m)$ das Relationensystem $\mathcal{R}_1 = (F_1, B, n_1, m_1)$ mit $n_1 < n$, $m_1 = n_1 + (m-n)$ und $F_1 \subset F$ sowie, falls A regulär ist, Det $\Lambda(A) =$ Det $\Lambda(B)$.	(4)
Versuche, zwei reguläre Teilmatrizen $B_1, B_2 \in \mathbb{Z}^{n_1 \times n_1}$ von B auszuwählen. Falls dies gelingt, d.h. falls B regulär ist und mindestens (n_1+1) von Null verschiedene Spalten enthält, berechne mit Hilfe einer modifizierten Form der modularen Gauß-Elimination simultan die Determinanten d_1 und d_2 von B_1 und B_2 und setze $d \leftarrow \gcd(d', d'')$. Andernfalls setze $d_1 \leftarrow 0$.	(5)
BREAKIF $d \neq 0$.	(6)
Generiere eine geeignete Anzahl k neuer Relationen $v_1, \ldots v_k$ über F_1 und füge diese als zusätzliche Spalten an B an. Erhöhe m_1 um k.	(7)
Berechne mit Hilfe modularer Hermite-Reduktion eine Basis H des Gitters $\Lambda(B)$ in Hermite-Normalform. Verwende dabei d als Modul.	(8)
Setze $h \leftarrow \det H$.	(9)
BREAKIF $h < 2h^*$ (d.h. die Spalten der Matrix H erzeugen das volle Relationengitter über F_1).	(10)
Generiere eine neue Relation v über F_1.	(11)
Setze $H' \leftarrow H$ und füge v als zusätzliche Spalte an H' an. Berechne eine Basis H des Gitters $\Lambda(H')$ in Hermite-Normalform.	(12)
Berechne die Smith-Normalform S von H und bestimme so die Struktur der Klassengruppe und ein minimales Erzeugendensystem.	(13)

In den folgenden fünf Kapiteln werden, wie bereits erwähnt, die einzelnen Schritte des Algorithmus CLASSGROUP ausführlich beschrieben. Die Beschreibung wird durch Tabellen mit umfangreichem Datenmaterial ergänzt. Die Daten stammen aus Anwendungen des Algorithmus CLASSGROUP für 36 Diskriminanten mit 11 bis 46 Dezimalstellen. Sie zeigen, wie verschiedene wichtige Parameter des Algorithmus von der Größe der Diskriminante abhängen.

Der Algorithmus CLASSGROUP wurde in der Programmiersprache C auf Rechnern des Typs SparcStation unter dem Betriebssystem SunOS 4.1 implementiert und an vielen Beispielen erfolgreich getestet. Auf die technischen Details der Implementierung wird in dieser Arbeit nicht eingegangen. Es sei lediglich angemerkt, daß über die hier dargestellten Verfahren hinaus eine Reihe weiterer Algorithmen implementiert werden mußten, so z.B. eine effiziente Arithmetik zum Rechnen mit beliebig großen ganzen Zahlen.

Kapitel 7

Die Initialisierung

In diesem Kapitel beschreiben wir die Initialisierungsschritte (1) und (2) des Algorithmus CLASSGROUP. Hier ist für vorgegebene Diskriminante $\Delta \in \mathcal{D}^-$ zum einen eine Approximation $h^* \in I\!R$ zu berechnen, so daß

$$h^* < h(\Delta) < 2h^* \tag{7.1}$$

gilt, zum anderen ist eine geeignete Formenbasis F für die Klassengruppe $Cl(\Delta)$ zu bestimmen.

7.1 Approximation der Klassenzahl

Wir führen alle Überlegungen für festes $\Delta \in \mathcal{D}^-$ durch. Dabei verzichten wir in vielen Fällen auf die explizite Angabe der Abhängigkeit der verwendeten Größen von Δ, d.h. wir schreiben z.B. für die Eulerfaktoren $E(p)$ anstelle von $E(\Delta, p)$.

Die Berechnung von h^* erfolgt, ähnlich wie in Abschnitt 4.3 angegeben, durch die Auswertung endlich vieler Terme der analytischen Klassenzahlformel (4.4). Wir berechnen also für ein geeignetes $Q \in I\!N$ eine Approximation der Form

$$\tilde{h}(Q) = \frac{\sqrt{|\Delta|}}{\pi} \cdot \prod_{\substack{p \in I\!P, \\ p \leq Q}} E(p) \tag{7.2}$$

und setzen dann

$$h^* = \mu \cdot \tilde{h}(Q) \tag{7.3}$$

für einen geeigneten Parameter $\mu \in I\!R$ mit $\frac{1}{2} < \mu < 1$. Dazu müssen Q und μ so gewählt werden, daß h^* die Forderung (7.1) erfüllt.

In Abschnitt 4.3 wurde bereits erläutert, daß man dazu unter Verwendung der Konstanten c_4 aus Korollar 4.8 einen Wert $Q > c_4 \log^2 |\Delta|$ wählen und $\mu = \frac{3}{4}$ setzen kann. Dieses Ergebnis reicht jedoch für die Praxis nicht aus, da in Korollar 4.8 nur eine Aussage über die Existenz der Konstanten c_4 gemacht wird, nicht jedoch angegeben wird, wie man diese Konstante berechnet. Wir untersuchen daher in diesem

Abschnitt, wie man in Abhängigkeit von Δ eine Schranke $Q_0 \in I\!N$ berechnen kann, so daß für $Q > Q_0$ und geeignet gewähltes μ die nach (7.3) berechnete Zahl h^* die Forderung (7.1) erfüllt.

Wir bezeichnen den (multiplikativen) Fehler, mit dem die Approximation $\tilde{h}(Q)$ behaftet ist, mit $T(Q)$, d.h. wir setzen

$$T(Q) = \frac{h(\Delta)}{\tilde{h}(Q)}. \tag{7.4}$$

Offensichtlich ist für festes μ die Forderung (7.1) genau dann erfüllt, wenn

$$\mu\tilde{h}(Q) < h(\Delta) < 2\mu\tilde{h}(Q)$$

ist, wenn also

$$\mu < T(Q) < 2\mu \tag{7.5}$$

gilt. Wir werden später eine obere Schranke für $|\log T(Q)|$ angeben und beschäftigen uns nun zunächst mit der Wahl eines geeigneten Parameters μ. Die Forderung (7.5) ist offenbar äquivalent zu der Forderung

$$\log \mu < \log T(Q) < \log(2\mu),$$

für deren Gültigkeit die Bedingung

$$|\log T(Q)| < M = \min\{-\log \mu,\ \log(2\mu)\}$$

erfüllt sein muß. Wir wählen nun μ so, daß M maximal wird. Dies ist der Fall für $\log(2\mu) = -\log \mu$, also für

$$\mu = \frac{1}{2}\sqrt{2}. \tag{7.6}$$

Bei dieser Wahl von μ ist die Forderung (7.5) erfüllt, wenn

$$|\log T(Q)| < \log\sqrt{2} \tag{7.7}$$

ist.

Wir leiten nun, wie angekündigt, eine obere Schranke für $|\log T(Q)|$ in Abhängigkeit von Q und Δ her. Es handelt sich dabei um die Spezialisierung eines allgemeineren Resultates von Buchmann und Williams [9] für den Fall imaginär-quadratischer Klassenzahlen.

Mit Hilfe von (4.4), (7.2) und (7.4) folgt zunächst, daß

$$T(Q) = \prod_{\substack{p \in \boldsymbol{P}, \\ p > Q}} E(p) \tag{7.8}$$

ist.

Offensichtlich kann man die Eulerfaktoren $E(p)$ auch in der Form

$$E(p) = \left(1 + \frac{a(p)}{p}\right)^{-1} \tag{7.9}$$

schreiben, wobei

$$a(p) = -\left(\frac{\Delta}{p}\right) \in \{-1, 0, 1\}$$

zu setzen ist. Für alle $p \in I\!P$ ist offensichtlich $\left|\frac{a(p)}{p}\right| < 1$ und daher $0 < E(p) < 2$. Für $p \mid \Delta$ ist $a(p) = 0$ und daher $E(p) = 1$. Wir müssen also in dem Produkt (7.8) nur solche Primzahlen $p > Q$ betrachten, die die Diskriminante nicht teilen. Dazu setzen wir

$$P^* = \{p \in I\!P \;:\; p > Q,\; p \nmid \Delta\}$$

und erhalten dann aus (7.8) und (7.9)

$$T(Q) = \prod_{p \in P^*} \left(1 + \frac{a(p)}{p}\right)^{-1}. \tag{7.10}$$

Da $E(p) > 0$ ist, folgt hieraus

$$-\log T(Q) = \sum_{p \in P^*} \log\left(1 + \frac{a(p)}{p}\right). \tag{7.11}$$

Da außerdem $\left|\frac{a(p)}{p}\right| < 1$ ist, können wir die Terme $\left(1 + \frac{a(p)}{p}\right)$ in Potenzreihen entwickeln und erhalten

$$\begin{aligned} -\log T(Q) &= \sum_{p \in P^*} \sum_{j=1}^{\infty} \left(\frac{a(p)}{p}\right)^j \cdot \frac{(-1)^{j+1}}{j} & (7.12)\\ &= \sum_{p \in P^*} \frac{a(p)}{p} + \sum_{p \in P^*} \sum_{j=2}^{\infty} \left(\frac{a(p)}{p}\right)^j \cdot \frac{(-1)^{j+1}}{j}. & (7.13) \end{aligned}$$

Die letzte Umformung ist nach dem Doppelreihensatz von Cauchy (vgl. [19], S. 258) gerechtfertigt, da offensichtlich die Reihe (7.12) absolut konvergent ist. Wir setzen nun

$$\begin{aligned} S_1(Q) &= \sum_{p \in P^*} \frac{a(p)}{p}, \\ S_2(Q) &= \sum_{p \in P^*} \sum_{j=2}^{\infty} \left(\frac{a(p)}{p}\right)^j \cdot \frac{(-1)^{j+1}}{j} \end{aligned}$$

und geben die beiden folgenden Lemmata an:

Lemma 7.1 *(GRH) Für $Q \geq 2$ ist*

$$|S_1(Q)| \leq \frac{4 + 3\log Q}{\sqrt{Q}} \cdot \tilde{C}(Q),$$

wobei

$$\tilde{C}(Q) = \log|\Delta| \left(\frac{1}{\pi \log Q} + \frac{5.3}{\log^2 Q}\right) + \left(\frac{1}{\pi} + \frac{4}{\log Q}\right).$$

Der Beweis dieses Lemmas ergibt sich unmittelbar aus der Darstellung in [9], wenn man die dortigen Resultate auf den Fall imaginär-quadratischer Klassengruppen spezialisiert. Wir geben ihn daher hier nicht gesondert an.

Lemma 7.2 *Für $Q \in \mathbb{N}$ ist*

$$|S_2(Q)| \leq \frac{2}{Q}.$$

Der Beweis für dieses Lemma ergibt sich ebenfalls aus der Darstellung in [9]. Wir geben ihn hier dennoch an, da er für den hier vorliegenden Fall stark verändert wird.

Beweis: Zunächst stellen wir fest, daß für $p \in \mathbb{P}$ und $j \geq 2$ die Abschätzung

$$\left| \left(\frac{a(p)}{p}\right)^j \cdot \frac{(-1)^{j+1}}{j} \right| \leq \frac{|a(p)|^j}{p^j} \leq \left(\frac{1}{p}\right)^j$$

gilt, wobei für die letzte Ungleichung die Tatsache $|a(p)| = 1$ zu beachten ist. Weiterhin ergibt sich mit Hilfe der geometrischen Reihe

$$\sum_{j=2}^{\infty} \left(\frac{1}{p}\right)^j = \frac{1}{p^2(1-\frac{1}{p})} = \frac{1}{p^2} \cdot \frac{p}{p-1} \leq \frac{2}{p^2}.$$

Da für $Q \in \mathbb{N}$

$$\sum_{p>Q} \frac{1}{p^2} \leq \frac{1}{Q}$$

gilt, folgt die Aussage von Lemma 7.2. ■

Aus Geichung (7.13) ergibt sich unmittelbar

$$|\log T(Q)| \leq |S_1(Q)| + |S_2(Q)|,$$

und damit ist der folgende Satz bewiesen.

Satz 7.3 *(GRH) Für $Q \geq 2$ ist*

$$|\log T(Q)| \leq C(Q) = \frac{4 + 3\log Q}{\sqrt{Q}} \cdot \tilde{C}(Q) + \frac{2}{Q}, \tag{7.14}$$

wobei

$$\tilde{C}(Q) = \log|\Delta| \left(\frac{1}{\pi \log Q} + \frac{5.3}{\log^2 Q}\right) + \left(\frac{1}{\pi} + \frac{4}{\log Q}\right). \tag{7.15}$$

Offensichtlich ist für festes Δ die Funktion $C(Q)$ streng monoton fallend. Insofern läßt sich mit Hilfe dieses Satzes für vorgegebenes $\Delta \in \mathcal{D}^-$ eine Zahl Q_0 bestimmen, so daß für $Q > Q_0$ die Ungleichung

$$C(Q) < \log \sqrt{2} \tag{7.16}$$

gilt und damit die Forderung (7.7) erfüllt ist. Nun ist jedoch die Ungleichung (7.16) nicht geschlossen nach Q auflösbar, so daß in der Praxis Q_0 für jeden Wert von Δ mit Hilfe einer Intervallschachtelung bestimmt wird.

Wir geben in der folgenden Tabelle für einige Werte von Δ den Wert von $Q_0(\Delta)$ an. Um den Zusammenhang zur Aussage von Korollar 4.8 deutlich zu machen, geben wir Q_0 auch als Vielfaches von $\log^2|\Delta|$ an.

Tabelle 7.4 (Parameter für die Approximation der Klassenzahl)

$\|\Delta\|$	$Q_0(\Delta)$
$4 \cdot 10^{10}+4$	$68195 = 115 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{11}+4$	$77056 = 109 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{12}+4$	$86375 = 104 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{13}+4$	$96145 = 99 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{14}+4$	$106364 = 95 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{15}+4$	$117026 = 92 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{16}+4$	$128129 = 89 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{17}+4$	$139667 = 86 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{18}+4$	$151638 = 84 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{19}+4$	$164038 = 82 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{20}+4$	$176864 = 80 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{21}+4$	$190114 = 78 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{22}+4$	$203785 = 76 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{23}+4$	$217874 = 75 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{24}+4$	$232378 = 73 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{25}+4$	$247297 = 72 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{26}+4$	$262626 = 71 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{27}+4$	$278365 = 70 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{28}+4$	$294511 = 69 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{29}+4$	$311063 = 68 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{30}+4$	$328019 = 67 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{31}+4$	$345375 = 66 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{32}+4$	$363133 = 65 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{33}+4$	$381288 = 65 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{34}+4$	$399841 = 64 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{35}+4$	$418790 = 63 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{36}+4$	$438132 = 63 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{37}+4$	$457867 = 62 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{38}+4$	$477994 = 62 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{39}+4$	$498511 = 61 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{40}+4$	$519416 = 60 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{41}+4$	$540709 = 60 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{42}+4$	$562388 = 59 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{43}+4$	$584453 = 59 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{44}+4$	$606902 = 59 \cdot \log^2\|\Delta\|$
$4 \cdot 10^{45}+4$	$629733 = 58 \cdot \log^2\|\Delta\|$

Zur Berechnung der Näherung h^* mit Hilfe der Formeln (7.2) und (7.3) ist neben der Kenntnis des Wertes $Q_0(\Delta)$ auch die Berechnung der Euler-Faktoren

$$E(p) = \left(1 - \left(\frac{\Delta}{p}\right)\frac{1}{p}\right)^{-1} = \frac{p}{p - \left(\frac{\Delta}{p}\right)} \tag{7.17}$$

für jede Primzahl $p < Q_0(\Delta)$ erforderlich. Die Auswertung des Kroneckersymbols erfolgt dabei für $p \neq 2$ mit Algorithmus 4.18, für $p = 2$ wie zu Beginn von Abschnitt 4.5 beschrieben.

Da auch für die Konstruktion der Formenbasis F, die im nächsten Abschnitt diskutiert wird, die Berechnung von Kroneckersymbolen erforderlich ist, wird die Berechnung von h^* simultan dazu durchgeführt. Der entsprechende Algorithmus wird in Abschnitt 7.3 angegeben.

7.2 Konstruktion der Formenbasis

Wir erläutern in diesem Abschnitt die Berechnung einer geeigneten Formenbasis F für die Klassengruppe $Cl(\Delta)$. Auch dazu sei die Diskriminante $\Delta \in \mathcal{D}^-$ fest vorgegeben.

Nach den Ausführungen in Abschnitt 4.3 ist die Menge

$$F = \{f_i = (p_i, b_{p_i}) \ : \ f_i \text{ Primform in } \mathcal{QF}(\Delta),\ p_i < 6 \log^2 |\Delta|\}$$

eine Formenbasis für $Cl(\Delta)$. Zur Konstruktion dieser Formenbasis muß man für alle Primzahlen p unterhalb der Schranke $6 \cdot \log^2 |\Delta|$ zunächst den Wert des Kronecker-symbols $(\frac{\Delta}{p})$ bestimmen. Anschließend ist zu jeder der Primzahlen p mit $(\frac{\Delta}{p}) = 1$ die Zahl

$$b_p = \min\{b \in \mathbb{N} \ : \ b^2 \equiv \Delta \bmod (4p)\}$$

zu bestimmen. Dies geschieht mit Hilfe von Algorithmus 4.20, wie in Abschnitt 4.6 beschrieben wurde.

In Kapitel 6 wurde bereits erwähnt, daß wir noch einige weitere Formen in die Formenbasis aufnehmen, um die Erfolgswahrscheinlichkeit bei der Generierung der Relationen zu erhöhen. Diese Erweiterung der Formenbasis wird nun zunächst begründet.

Zur erfolgreichen Erzeugung einer Relation mit Hilfe der in Abschnitt 4.4 beschriebenen Idee ist es notwendig, daß die Form $g = (a, b) \in \mathcal{QF}(\Delta)$, die man zuvor durch Berechnung eines Potenzproduktes und anschließende Reduktion bestimmt hat, vollständig über der vorgegebenen Formenbasis F zerfällt. Die Zerlegung der Form g wird mit Hilfe von Satz 4.5 vorgenommen, der jedoch nur angewandt werden kann, wenn $\gcd(a, \Delta) = 1$ ist.

Wir wollen jedoch auch dann in der Lage sein, eine Zerlegung der Form g zu finden, wenn $\gcd(a, \Delta) \neq 1$ ist. Dazu formulieren wir in diesem Abschnitt eine erweiterte Fassung des Satzes 4.5. Um diesen neuen Satz sinnvoll anwenden zu können, ist die oben erwähnte Aufnahme zusätzlicher Formen in die Formenbasis notwendig.

Wir zeigen zunächst, daß das Auftreten reduzierter Formen (a, b) mit $\gcd(a, \Delta) \neq 1$ tatsächlich möglich ist. Dazu definieren wir den Begriff des einfachen Primteilers der Diskriminante.

Definition 7.5 *Es sei $\Delta \in \mathcal{D}^-$. Eine Primzahl p heißt* **einfacher Primteiler** *von Δ, wenn*

$$p \ | \ \Delta/4, \qquad p^2 \nmid \Delta/4.$$

Offensichtlich ist eine Primzahl $p \neq 2$ genau dann ein einfacher Primteiler von Δ, wenn $p \mid \Delta$, aber $p^2 \nmid \Delta$; für $p \neq 2$ entspricht diese Definition also der intuitiven Anschauung des Begriffes "einfacher Primteiler". Die Zahl 2 ist dagegen genau dann einfacher Primteiler von Δ, wenn $\Delta \equiv 8,\ 12 \bmod 16$ ist.

Satz 7.6 *Es sei $\Delta \in \mathcal{D}^-$ und $p \in \mathbb{P}$ einfacher Primteiler von Δ. Dann gilt: Für jede Form $(a, b) \in \mathcal{QF}(\Delta)$ mit $gcd(a, p) = 1$ gibt es ein $b' \in \mathbb{Z}$, so daß (ap, b') ebenfalls eine Form in $\mathcal{QF}(\Delta)$ ist.*

Beweis: Wir betrachten zunächst den Fall $p \neq 2$. Wir wählen $b' \in \mathbb{Z}$ mit

$$\begin{aligned} b' &\equiv 0 \bmod p, \\ b' &\equiv b \bmod (2a). \end{aligned}$$

Ein solches b' existiert nach dem Chinesischen Restsatz (Satz 3.33), da $\gcd(a, p) = 1$ ist.

Da $f = (a, b) \in \mathcal{QF}(\Delta)$ ist, gilt $b^2 - \Delta \equiv 0 \bmod (4a)$. Bei obiger Wahl von b' folgt daraus sofort, daß auch $b'^2 - \Delta \equiv 0 \bmod (4ap)$ ist. Es existiert also eine Form $f' = (ap, b')$, die jedoch nicht notwendigerweise primitiv ist. Um die Primitivität der Form f' nachzuweisen, müssen wir zeigen, daß $\gcd(ap, b', c') = 1$ ist, wobei

$$c' = \frac{b'^2 - \Delta}{4ap}$$

der dritte Koeffizient der Form f' ist. Wir setzen dazu zunächst $r = \gcd(a, b)$. Nach Wahl von b' folgt dann, daß $\gcd(ap, b') = rp$ ist. Es bleibt also zu zeigen, daß $\gcd(rp, c') = 1$ ist.

Es sei nun

$$c = \frac{b^2 - \Delta}{4a}$$

der dritte Koeffizient der Form f. Dann gilt

$$p \cdot c' = \frac{(b + 2\nu a)^2 - \Delta}{4a} = \frac{b^2 - \Delta}{4a} + \frac{4\nu ab + 4\nu^2 a^2}{4a} = c + \nu b + \nu^2 a$$

für ein geeignetes $\nu \in \mathbb{Z}$. Nun ist nach Voraussetzung die Form f primitiv, d.h. es ist $\gcd(r, c) = 1$. Da aber $r \mid (\nu b + \nu^2 a)$, ist $\gcd(r, c'p) = 1$, also auch $\gcd(r, c') = 1$. Andererseits gilt

$$4a \cdot c' = \frac{(\mu p)^2 - \Delta}{p} = \mu^2 p - (\Delta/p)$$

für ein geeignetes $\mu \in \mathbb{Z}$. Da p einfacher Primteiler von Δ ist, folgt $p \nmid 4ac'$, also $p \nmid c'$. Damit ist die Primitivität der Form f' bewiesen.

Wir betrachten nun den Fall $p = 2$. Hier wählen wir $b' \in \mathbb{Z}$ mit

$$\begin{aligned} b' &\equiv b \bmod a && \text{und} \\ b' &\equiv 0 \bmod 4 && \text{falls } \Delta \equiv 8 \bmod 16 \text{ bzw.} \\ b' &\equiv 2 \bmod 16 && \text{falls } \Delta \equiv 12 \bmod 16. \end{aligned}$$

Da $\gcd(a, 2) = 1$ ist, folgt auch hier die Existenz einer solchen Zahl b' aus dem Chinesischen Restsatz. Da b und b' gerade sind und a ungerade ist, erfüllt b' sogar die Kongruenz

$$b' \equiv b \bmod (2a).$$

Die weitere Argumentation entspricht der für den Fall $p \neq 2$, das zentrale Argument für den Beweis der Primitivität ist die Tatsache, daß $b'^2 - \Delta$ zwar durch 8, nicht aber durch 16 teilbar ist. ■

Zu jedem einfachen Primteiler p der Diskriminante Δ und jeder Form $(a,b) \in \mathcal{QF}(\Delta)$ mit $\gcd(a,p) = 1$ gibt es also eine Form $(a',b') \in \mathcal{QF}(\Delta)$ mit $\gcd(a',p) = p$. Die so konstruierten Formen sind natürlich nicht in allen Fällen reduziert, selbst wenn die Form (a,b) reduziert ist. Trotzdem zeigt aber diese Betrachtung, daß für jeden einfachen Primteiler p von Δ einige reduzierte Formen existieren, deren erster Koeffizient nicht teilerfremd zur Diskriminante ist. Es ist also sinnvoll, eine Möglichkeit zur Faktorisierung solcher Formen zu suchen, denn man kann erwarten, daß dadurch die Erfolgswahrscheinlichkeit für die Generierung der Relationen deutlich ansteigt. Dies gilt insbesondere dann, wenn die Diskriminante viele kleine Primteiler hat.

In diesem Zusammenhang ist es auch von Interesse, ob Formen $(a,b) \in \mathcal{QF}(\Delta)$ existieren, für die der Koeffizient a von einem einfachen Primteiler der Diskriminante zu einer höheren als der ersten Potenz geteilt wird. Diese Frage beantwortet der folgende Satz:

Satz 7.7 *Es sei $\Delta \in \mathcal{D}^-$ und $p \in \mathbb{P}$ ein einfacher Primteiler von Δ. Dann existiert keine Form $(a,b) \in \mathcal{QF}(\Delta)$ mit $p^2 \mid a$.*

Beweis: Wir betrachten wiederum zunächst den Fall $p \neq 2$ und nehmen an, es gäbe eine Form (a,b) mit $p^2 \mid a$. Dann wäre p^2 ein Teiler von $b^2 - \Delta$. Da $p \mid \Delta$, folgt $p \mid b^2$ und daher $p^2 \mid b^2$. Da aber $p^2 \nmid \Delta$, kann p^2 kein Teiler von $b^2 - \Delta$ sein.

Wir betrachten nun den Fall $p = 2$, nehmen also an, es gäbe eine Form (a,b) mit $4 \mid a$. Dann wäre 16 ein Teiler von $b^2 - \Delta$. Um dies zu ermöglichen, müßte $b^2 \equiv \Delta \bmod 16$ sein. Dies ist aber nicht möglich, da 8 und 12 keine quadratischen Reste modulo 16 sind. ■

Wir erweitern nun den in Abschnitt 4.1 definierten Begriff der Primform auf einfache Primteiler der Diskriminante. Dazu benötigen wir den folgenden Satz:

Satz 7.8 *Es sei $\Delta \in \mathcal{D}^-$ und $p \in \mathbb{P}$ ein einfacher Primteiler von Δ. Es sei*

$$b_p = \begin{cases} 0 & \text{falls } \Delta \equiv 0 \bmod 4, \\ p & \text{falls } \Delta \equiv 1 \bmod 4. \end{cases},$$

falls $p \neq 2$, und

$$b_p = \begin{cases} 0 & \text{falls } \Delta \equiv 8 \bmod 16, \\ 2 & \text{falls } \Delta \equiv 12 \bmod 16. \end{cases},$$

falls $p = 2$. Dann gilt:

$$(p, b_p) \in \mathcal{QF}(\Delta).$$

Der Beweis dieses Satzes ergibt sich unmittelbar aus der Definition des Formenbegriffs, indem man nachrechnet, daß

$$c_p = \frac{b_p^2 - \Delta}{4p} \in \mathbb{Z}$$

ist und daß p kein Teiler von c_p ist.

Definition 7.9 *Es sei $\Delta \in \mathcal{D}^-$ und $p \in I\!P$ ein einfacher Primteiler von Δ. Die Form $f_p = (p, b_p)$ mit b_p wie in Satz 7.8 heißt* **ambige Primform.**

Ambige Primformen haben die folgende wichtige Eigenschaft:

Satz 7.10 *Es sei $\Delta \in \mathcal{D}^-$ und $(p, b_p) \in \mathcal{QF}(\Delta)$ eine ambige Primform. Dann gilt:*

$$[(p, b_p)]^2 = 1_{\mathcal{Cl}}.$$

Beweis: Wir verwenden die in Definition 2.4 genannten Formeln für die Produktbildung zweier Formen mit $a_1 = a_2 = p$ und $b_1 = b_2 = b_p$. Wir erhalten dann zunächst mit den Formeln (2.3) und (2.4) $d = p$ und $v_1 = 1$, $v_2 = w = 0$. Daraus ergibt sich die Produktform (a_3, b_3) mit $a_3 = 1$ und $b_3 = b$, die offensichtlich in der Klasse $1_{\mathcal{Cl}}$ liegt. ■

Im allgemeinen werden alle Formen, deren Quadrat in der Einsklasse liegt, als ambige Formen bezeichnet. Ambige Formen sind auch im Zusammenhang mit der Faktorisierung großer ganzer Zahlen von Bedeutung (vgl. [38], [39]). Wir gehen darauf jedoch nicht näher ein.

Wir geben nun an, wie man eine Form $g = (a, b)$, bei der a von einem einfachen Primteiler p der Diskriminante geteilt wird, so verändern kann, daß der erste Koeffizient der entstehenden Form nicht mehr von p geteilt wird.

Satz 7.11 *Es sei $\Delta \in \mathcal{D}^-$ und p einfacher Teiler von Δ. Ist $(a, b) \in \mathcal{QF}(\Delta)$ mit $p \mid a$, so gilt:*

(i)
$$(a, b) \cdot (p, b_p) = (a/p, b),$$

(ii)
$$[(a, b)] = [(a/p, b)] \cdot [(p, b_p)].$$

Beweis: (i) Wir verwenden wiederum Definition 2.4. Mit $a_1 = a$, $b_1 = b$, $a_2 = p$ und $b_2 = b_p$ ergibt sich mit den Formeln (2.3) und (2.4) $d = p$, $v_1 = w = 0$ und $v_2 = 1$. Daraus folgt, daß $a_3 = a \cdot p/p^2 = a/p$ und $b_3 = b \cdot p/p = b$ die Koeffizienten der Produktform (a_3, b_3) sind.

(ii) Aus (i) folgt $[(a, b)] \cdot [(p, b_p)] = [(a/p, b)]$, also $[(a, b)] = [(a/p, b)] \cdot [(p, b_p)]^{-1}$. Nach Satz 7.10 hat die Klasse $[(p, b_p)]$ die Ordnung 2, also ist $[(p, b_p)]^{-1} = [(p, b_p)]$ und damit ist (ii) bewiesen. ■

Man beachte, daß die in Satz 7.11 unter (ii) genannte Gleichheit im allgemeinen falsch ist, wenn man sie statt für Äquivalenzklassen für die entsprechenden Formen betrachtet.

Wir formulieren nun die bereits zu Beginn dieses Abschnittes erwähnte Erweiterung des Satzes 4.5. Mit Hilfe dieser Erweiterung ist es möglich, eine Form $g = (a, b) \in \mathcal{QF}(\Delta)$ auch dann über einer geeignet gewählten Formenbasis zu zerlegen, wenn $\gcd(a, \Delta) \neq 1$ ist. Vorausgesetzt wird lediglich, daß $\gcd(a, \Delta)$ nur aus einfachen Primteilern der Diskriminante besteht.

Satz 7.12 *Es sei $\Delta \in \mathcal{D}^-$ und $(a,b) \in \mathcal{QF}(\Delta)$, so daß $\gcd(a,\Delta)$ nur aus einfachen Primteilern der Diskriminante zusammengesetzt ist. Es sei*

$$a = \prod_{i=1}^{n} p_i^{e_i}$$

mit geeignetem $n \in \mathbb{N}$ sowie $e_1, \ldots, e_n \in \mathbb{N}$ und $p_1, \ldots, p_n \in \mathbb{P}$ die Primfaktorzerlegung von a. Es seien $I_A \subset [\![1,n]\!]$ und $I_R = [\![1,n]\!] \setminus I_A$, so daß

$$\begin{array}{ll} p_i \nmid \Delta & (i \in I_R), \\ p_i \text{ einfacher Primteiler von } \Delta & (i \in I_A). \end{array}$$

Für $1 \le i \le n$ seien ganze Zahlen b_{p_i} so gewählt, daß (p_i, b_{p_i}) gewöhnliche Primformen $(i \in I_R)$ bzw. ambige Primformen $(i \in I_A)$ in $\mathcal{QF}(\Delta)$ sind. Dann gilt:

(i)
$$\left(\tfrac{\Delta}{p_i}\right) = 1 \qquad (i \in I_R).$$

(ii)
$$b \equiv \pm b_{p_i} \bmod (2p_i) \qquad (i \in I_R).$$

(iii)
$$e_i = 1 \qquad (i \in I_A).$$

(iv)
$$[(a,b)] = \prod_{i \in I_R} [(p_i, b_{p_i})]^{\pm e_i} \cdot \prod_{i \in I_A} [(p_i, b_{p_i})], \tag{7.18}$$

wobei für $i \in I_R$ das positive Vorzeichen des Exponenten e_i jeweils genau dann gilt, wenn $b \equiv +b_{p_i} \bmod (2p_i)$ ist.

Beweis: Die Aussage (iii) folgt unmittelbar aus Satz 7.7. Mit Hilfe mehrfacher Anwendung des Satzes 7.11 kann man schließen, daß

$$[(a,b)] = [(a',b)] \cdot \prod_{i \in I_A} [(p_i, b_{p_i})]$$

ist, wobei

$$a' = \prod_{i \in I_R} p_i^{e_i}$$

ist, also insbesondere $\gcd(a', \Delta) = 1$. Mit Hilfe von Satz 4.5 folgen daraus die Aussagen (i), (ii) und (iv). ■

Um Satz 7.12 sinnvoll anwenden zu können, müssen auch ambige Primformen in der Menge F enthalten sein. Wir benutzen daher für den Algorithmus CLASSGROUP die Formenbasis

$$F = \{ f_i = (p_i, b_{p_i}) \ : \ f_i \text{ gewöhnliche oder ambige Primform in } \mathcal{QF}(\Delta), \ p_i < N \}$$

mit $N = 6 \log^2 |\Delta|$.

Es stellt sich nun die Frage, ob man mit quadratischen oder höheren Teilern der Diskriminante in ähnlicher Weise verfahren kann. Bei genauer Untersuchung dieser

Frage stellt man fest, daß sich die Überlegungen dieses Abschnittes nicht in derselben Weise auf höhere Teiler der Diskriminante verallgemeinern lassen. Die Behandlung quadratischer Teiler der Diskriminante ist vielmehr wesentlich komplizierter (vgl. [11]). Die Berücksichtigung solcher Primteiler für die Formenbasis würde in der Praxis einen so großen Verwaltungsaufwand verursachen, daß insgesamt keine Effizienzverbesserung zu erwarten wäre.

Die folgende Tabelle zeigt für einige Werte von Δ die Auswirkungen der Aufnahme ambiger Primformen in die Formenbasis. In der zweiten und dritten Spalte wird die Gesamtzahl n der Elemente in der Formenbasis F und die Anzahl n_a der ambigen Formen in F angegeben. Hier wird deutlich, daß nur ein sehr kleiner Anteil der Primformen in F ambige ist. Die Erweiterung der Formenbasis durch die Hinzunahme der ambigen Formen ist also tatsächlich nur sehr geringfügig. In der vierten Spalte wird der Anteil t_a der Relationen angegeben, an denen ambige Primformen beteiligt sind. Hier wird deutlich, daß die Hinzunahme der ambigen Primformen zur Formenbasis tatsächlich zu einer wesentlichen Erhöhung der Erfolgswahrscheinlichkeit bei der Generierung der Relationen führt.

Tabelle 7.13 (Ambige Primformen in der Formenbasis)

$\|\Delta\|$	n	n_a	t_a
$4 \cdot 10^{10} + 4$	249	3	37.4%
$4 \cdot 10^{11} + 4$	300	3	40.0%
$4 \cdot 10^{12} + 4$	339	3	38.7%
$4 \cdot 10^{13} + 4$	391	3	41.4%
$4 \cdot 10^{14} + 4$	458	4	38.4%
$4 \cdot 10^{15} + 4$	483	7	41.4%
$4 \cdot 10^{16} + 4$	556	5	38.4%
$4 \cdot 10^{17} + 4$	613	4	58.8%
$4 \cdot 10^{18} + 4$	649	3	36.0%
$4 \cdot 10^{19} + 4$	708	2	48.9%
$4 \cdot 10^{20} + 4$	796	3	37.2%
$4 \cdot 10^{21} + 4$	896	5	48.0%
$4 \cdot 10^{22} + 4$	953	3	44.2%
$4 \cdot 10^{23} + 4$	1001	5	50.4%
$4 \cdot 10^{24} + 4$	1131	2	47.0%
$4 \cdot 10^{25} + 4$	1181	5	47.5%
$4 \cdot 10^{26} + 4$	1330	3	39.3%
$4 \cdot 10^{27} + 4$	1304	5	66.2%

$\|\Delta\|$	n	n_a	t_a
$4 \cdot 10^{28} + 4$	1441	4	41.9%
$4 \cdot 10^{29} + 4$	1511	3	51.6%
$4 \cdot 10^{30} + 4$	1604	6	47.2%
$4 \cdot 10^{31} + 4$	1754	2	48.2%
$4 \cdot 10^{32} + 4$	1860	2	39.2%
$4 \cdot 10^{33} + 4$	1933	6	63.7%
$4 \cdot 10^{34} + 4$	2010	2	42.5%
$4 \cdot 10^{35} + 4$	2090	3	48.2%
$4 \cdot 10^{36} + 4$	2238	4	44.0%
$4 \cdot 10^{37} + 4$	2339	3	50.0%
$4 \cdot 10^{38} + 4$	2439	2	42.3%
$4 \cdot 10^{39} + 4$	2483	6	62.0%
$4 \cdot 10^{40} + 4$	2695	2	46.1%
$4 \cdot 10^{41} + 4$	2761	2	50.3%
$4 \cdot 10^{42} + 4$	2939	5	48.7%
$4 \cdot 10^{43} + 4$	3112	2	50.0%
$4 \cdot 10^{44} + 4$	3216	4	44.8%
$4 \cdot 10^{45} + 4$	3314	11	70.1%

Man beachte, daß die Formenbasis F für $|\Delta| = 4 \cdot 10^{27} + 4$ weniger Elemente enthält als für $|\Delta| = 4 \cdot 10^{26} + 4$. Die Anzahl der Elemente in der Formenbasis muß also bei steigender Diskriminante nicht notwendigerweise auch zunehmen.

7.3 Der Algorithmus zur Initialisierung

Wir geben in diesem Abschnitt den Algorithmus zur Initialisierung der Klassengruppenberechnung an. Wie bereits erwähnt, führen wir die Berechnung von h^* und die Konstruktion der Formenbasis F gleichzeitig durch, da für beide Aufgaben die Berechnung derselben Kroneckersymbole erforderlich ist.

Der Algorithmus erhält als Eingabe eine Diskriminante $\Delta \in \mathcal{D}^-$. Weiterhin wird die Existenz einer (genügend langen) Liste von Primzahlen vorausgesetzt, in der die Primzahlen $p > 2$ der Größe nach geordnet enthalten sind. Der Algorithmus berechnet folgende Daten:

- die Schranke h^* mit $h^* < h(\Delta) < 2h^*$,
- die Formenbasis F,
- die Gesamtzahl n der Elemente in F,
- die Anzahl $k \leq n$ der gewöhnlichen Primformen in F,
- die Menge M der mehrfachen Primteiler der Diskriminante unterhalb von $6 \log^2 |\Delta|$.

Die Formenbasis F soll nach den Ausführungen im vorigen Abschnitt alle gewöhnlichen und ambigen Primformen $f_p \in \mathcal{QF}(\Delta)$ mit $p < 6 \log^2 |\Delta|$ enthalten. Sie wird realisiert durch eine Liste, in der zunächst die gewöhnlichen Primformen f_p geordnet nach der Größe von p und dann die ambigen Primformen f_p geordnet nach der Größe von p enthalten sind. Die einzelnen Primformen f_p sind dabei durch Angabe der Paare (p, f_p) repräsentiert.

Wir berechnen also die Formenbasis F in der Form

$$F = \{f_1, \dots, f_n\}$$

mit

$$f_i = (p_i, b_{p_i}) \qquad (1 \leq i \leq n),$$

so daß

- f_i gewöhnliche Primform in $\mathcal{QF}(\Delta)$ für $1 \leq i \leq k$,
- f_i ambige Primform in $\mathcal{QF}(\Delta)$ für $k < i \leq n$,
- $p_i < p_{i+1}$ für $1 \leq i < k$ und für $k < i < n$.

Dieser Aufbau der Formenbasis wird im Algorithmus zur Generierung der Relationen vorausgesetzt.

Wir formulieren nun den Algorithmus zur Initialisierung. Man beachte, daß die ambigen Primformen zunächst in einer Hilfsliste F' zwischengespeichert werden, die später an die Liste F der gewöhnlichen Primformen angehängt wird. Die Anzahl der Elemente in F' wird mit n' bezeichnet. Aus technischen Gründen wird ferner die Behandlung der Primzahl $p = 2$ als Teilalgorithmus formuliert.

Algorithmus 7.14 (Initialisierung der Klassengruppenberechnung)

Eingabe: Diskriminante $\Delta \in \mathcal{D}^-$
Ausgabe: h^*, F, n, k und M wie oben angegeben

Initialisiere h^*, F, k, F', n' und M gemäß den Ergebnissen für die Primzahl $p = 2$ (vgl. Algorithmus 7.15).				(1)
Setze $g_1 \leftarrow \lfloor 6 \cdot \log^2 \lvert\Delta\rvert \rfloor$ (Primzahlgrenze für die Berechnung der Formenbasis).				(2)
Setze $g_2 \leftarrow Q_0(\Delta)$ gemäß Abschnitt 7.1 (Primzahlgrenze für die Approximation der Klassenzahl).				(3)
Lies die erste Primzahl p aus der Liste der Primzahlen ($p = 3$).				(4)
WHILE $p \leq g_1$				(5)
	Berechne $J = (\frac{\Delta}{p})$.			(6)
	Setze $h^* \leftarrow h^* \cdot \frac{p}{p-J}$.			(7)
	IF $J = 0$			(8)
	THEN	IF $p^2 \nmid \Delta$ (p ist einfacher Teiler von Δ)		(9)
		THEN	Berechne die ambige Primform $f = (p, b_p)$.	(10)
			Setze $F' \leftarrow F' \cup \{f\}$ und erhöhe n' um 1.	(11)
		ELSE	Setze $M \leftarrow M \cup \{p\}$.	(12)
	ELSE	IF J = 1		(13)
		THEN	Berechne die (gewöhnliche) Primform $f = (p, b_p)$.	(14)
			Setze $F \leftarrow F \cup \{f\}$ und erhöhe k um 1.	(15)
	Lies die nächste Primzahl p aus der Liste der Primzahlen.			(16)
WHILE $p \leq g_2$				(17)
	Berechne $J = (\frac{\Delta}{p})$.			(18)
	Setze $h^* \leftarrow h^* \cdot \frac{p}{p-J}$.			(19)
	Lies die nächste Primzahl p aus der Liste der Primzahlen.			(20)
Setze $h^* \leftarrow h^* \cdot \frac{\sqrt{\lvert\Delta\rvert}}{\pi\sqrt{2}}$.				(21)
Setze $F \leftarrow F \cup F'$ und $n \leftarrow k + n'$.				(22)

Algorithmus 7.15 (Initialisierung für die Primzahl $p = 2$)

Eingabe: Diskriminante $\Delta \in \mathcal{D}^-$
Ausgabe: h^*, F, k, F', n' und M nach Bearbeitung der Primzahl $p = 2$

Setze $F \leftarrow \emptyset,\ \ k \leftarrow 0,\ \ F' \leftarrow \emptyset,\ \ n' \leftarrow 0$ und $M \leftarrow \emptyset$.			(1)
IF $\Delta \equiv 0 \bmod 4$			(2)
THEN	Setze $h^* \leftarrow 1$.		(3)
	IF $\Delta \equiv 8, 12 \bmod 16$		(4)
	THEN	Berechne die ambige Primform $f = (2, b_2)$.	(5)
		Setze $F' \leftarrow F' \cup \{f\}$ und erhöhe n' um 1.	(6)
	ELSE	Setze $M \leftarrow M \cup \{2\}$.	(7)
ELSE	IF $\Delta \equiv 1 \bmod 8$		(8)
	THEN	Setze $h^* \leftarrow 2$.	(9)
		Berechne die (gewöhnliche) Primform $f = (2, b_2)$.	(10)
		Setze $F \leftarrow F \cup \{f\}$ und erhöhe k um 1.	(11)
	ELSE	Setze $h^* \leftarrow 2/3$.	(12)

Kapitel 8

Die Generierung der Relationen

In diesem Kapitel beschreiben wir Schritt (3) des Algorithmus CLASSGROUP, also die Generierung der Relationen. Dieser Schritt ist die entscheidende Phase im gesamten Algorithmus; für große Diskriminanten werden hier mehr als 95% der Gesamtlaufzeit verbraucht.

Der Algorithmus zur Generierung der Relationen wird zur besseren Übersicht in den Rahmenalgorithmus RELATIONEN und mehrere Teilalgorithmen aufgeteilt. Die einzelnen Algorithmen werden in den nächsten Abschnitten zunächst in einfachen Versionen angegeben und dann sukzessive weiterentwickelt.

Folgende Größen seien fest vorgegeben:

- die Diskriminante Δ,
- die Gesamtzahl n der Elemente in der Formenbasis F,
- die Anzahl k der gewöhnlichen Primformen in F,
- die Formenbasis $F = \{f_1, \ldots, f_n\}$, deren Elemente entsprechend den Anmerkungen in Abschnitt 7.3 numeriert sind, und damit implizit auch die Menge $F_p = \{p_1, \ldots, p_n\}$ der zu den Primformen aus F gehörenden Primzahlen,
- die Menge M der mehrfachen Primteiler der Diskriminante unterhalb von $6 \log^2 |\Delta|$.

8.1 Die Grundversion des Algorithmus RELATIONEN

Wir geben nun eine erste Version unseres Algorithmus zur Generierung der Relationen an. Diese Version orientiert sich an der Grundidee aus Algorithmus 4.11, berücksichtigt aber zusätzlich die in Abschnitt 6.2 erhobene Forderung, sofort $(n+5)$

Relationen zu bestimmen, sowie die Tatsache, daß die Formenbasis auch ambige Primformen enthält.

Algorithmus 8.1 (Rahmenalgorithmus RELATIONEN (1. Fassung))

Eingabe: Δ, F, F_p, n, k und M wie oben angegeben
Ausgabe: eine Matrix $A \in \mathbb{Z}^{n \times m}$ mit $m \geq n+5$, deren Spalten Relationen über F sind

Setze $m \leftarrow 1$.	(1)
WHILE $m \leq n+5$	(2)
Wähle einen Exponentenvektor $x = (x_1, \ldots, x_n) \in \mathbb{Z}^n$ und berechne die reduzierte Form $g = (a, b)$ in der Klasse von $\prod_{i=1}^{n} f_i^{x_i}$. (Teilalgorithmus POTENZPROD)	(3)
Stelle fest, ob a vollständig über F_p zerfällt und, falls ja, berechne den Zerlegungsvektor $y = (y_1, \ldots, y_n) \in \mathbb{Z}^n$ mit $a = \prod_{i=1}^{n} p_i^{y_i}$. Gib ferner den unzerlegbaren Anteil r von a an. (Teilalgorithmus FAKTOR)	(4)
UNTIL $r = 1$.	(5)
Berechne den Relationenvektor $w = (w_1, \ldots, w_n) \in \mathbb{Z}^n$. (Teilalgorithmus RELVEKTOR)	(6)
IF w ist nicht der Nullvektor	(7)
THEN Setze (Spalte) $A_m \leftarrow w$ und $m \leftarrow m+1$.	(8)

Der Rahmenalgorithmus hat neben dem Aufruf der Teilalgorithmen (Schritte (3), (4) und (6)) im wesentlichen die Aufgabe, die erzeugten Relationen zu zählen (Schritte (1), (2) und (7)). Der Algorithmus bricht ab, wenn $n+5$ Relationen vorhanden sind. In Schritt (7) wird der Zähler nur dann erhöht, wenn der gefundene Relationenvektor nicht der Nullvektor ist, da diese triviale Relation keine sinnvollen Informationen enthält und nicht in der Relationenmatrix A auftreten soll.

Der Teilalgorithmus POTENZPROD wird hier zunächst nicht angegeben; er wird im folgenden Abschnitt ausführlich besprochen.

Wir erläutern nun den Teilalgorithmus FAKTOR. Zunächst gehen wir dabei auf die Bedeutung des Parameters r ein, der in Schritt (5) des Algorithmus RELATIONEN ausgewertet wird. Mit Hilfe von r teilt der Algorithmus FAKTOR dem Algorithmus RELATIONEN mit, welches Ergebnis bei dem Versuch der Zerlegung von a über der Menge F_p erzielt wurde. Der Wert von r ist dabei wie folgt zu interpretieren:

- Ist $r = 1$, so war die vollständige Zerlegung von a möglich. Es ist dann

$$a = \prod_{i=1}^{n} p_i^{y_i}.$$

- Ist $r > 1$, so war die vollständige Zerlegung von a nicht möglich. r enthält in diesem Fall den nicht zerlegbaren Anteil von a, d.h. es gilt

$$a = r \cdot \prod_{i=1}^{n} p_i^{y_i},$$
$$p_i \nmid r \qquad (1 \le i \le n).$$

- Ist $r = 0$, so war die vollständige Zerlegung von a ebenfalls nicht möglich. In diesem Fall wurde keine vollständige Probedivision vorgenommen, so daß die Einträge des Vektors y keine sinnvollen Informationen enthalten.

In der ersten Fassung des Rahmenalgorithmus RELATIONEN werden in Schritt (5) nur die Fälle $r = 1$ und $r \neq 1$ unterschieden, der genaue Wert von r spielt im zweiten Fall zunächst keine Rolle. Aus Gründen der Konsistenz werden die Eigenschaften von r jedoch schon hier vollständig eingeführt.

Wir formulieren nun den Teilalgorithmus FAKTOR:

Algorithmus 8.2 (Teilalgorithmus FAKTOR (1. Fassung))

Eingabe: der Koeffizient $a \in \mathbb{N}$ einer reduzierten Form aus $\mathcal{QF}(\Delta)$ sowie F_p, n, k und M wie oben angegeben

Ausgabe: der zu a gehörende Zerlegungsvektor $y = (y_1, \ldots, y_n) \in \mathbb{Z}^n$ sowie r wie oben angegeben

IF a wird von einer der Primzahlen in der Menge M geteilt THEN setze $r \leftarrow 0$ und EXIT.	(1)
Setze $r \leftarrow a$.	(2)
FOR $i = 1$ TO n setze $y_i \leftarrow 0$.	(3)
FOR $i = k + 1$ TO n	(4)
IF $p_i \mid r$ THEN setze $r \leftarrow r/p_i$ und $y_i \leftarrow 1$.	(5)
FOR $i = 1$ TO k	(6)
WHILE $p_i \mid r$	(7)
Setze $r \leftarrow r/p_i$ und erhöhe y_i um 1.	(8)
IF $r = 1$ THEN EXIT.	(9)

In Schritt (1) wird zunächst untersucht, ob a von einem der Mehrfachteiler der Diskriminante geteilt wird, die während der Berechnung der Formenbasis in der Initialisierungsphase des Algorithmus CLASSGROUP entdeckt wurden. In diesem Fall ist eine Zerlegung von a über F_p nicht möglich. Daher wird $r = 0$ gesetzt, und der Algorithmus FAKTOR wird abgebrochen.

In den Schritten (4) bis (5) erfolgt die Probedivision durch die zu den ambigen Primformen aus F gehörenden Primzahlen. Diese Primzahlen können nach Satz 7.12 (iii) höchstens zur ersten Potenz in a aufgehen.

In den Schritten (6) bis (9) erfolgt dann die vollständige Probedivision durch alle übrigen Primzahlen aus F_p. In Schritt (9) wird der Algorithmus beendet, falls die Zerlegung bereits vollständig ist. Wenn der Algorithmus die Schleife (6) - (9) mit $i = n$ verläßt, ohne vorher in Schritt (9) beendet worden zu sein, so ist nach Durchführung aller Probedivisionen $r \neq 1$, d.h. a zerfällt nicht über F_p.

Wir geben nun den Teilalgorithmus RELVEKTOR an. In Schritt (2) wird für die gewöhnlichen Primformen das korrekte Vorzeichen des Eintrags im Formenzerlegungsvektor nach Satz 7.12 bestimmt.

Algorithmus 8.3 (Teilalgorithmus RELVEKTOR)

Eingabe: die Vektoren x und y, der Koeffizient b der zu zerlegenden Form g sowie F, n und k wie oben angegeben

Ausgabe: der Relationenvektor $w \in \mathbb{Z}^n$

FOR $i = 1$ TO k		(1)
	IF $y_i \neq 0$ und $b \equiv -b_{p_i} \bmod (2p_i)$ THEN setze $y_i \leftarrow -y_i$.	(2)
FOR $i = 1$ TO n setze $w_i \leftarrow x_i - y_i$.		(3)

In den folgenden Abschnitten werden nun die hier angegebenen Algorithmen schrittweise weiterentwickelt. Aus unseren Betrachtungen in Abschnitt 6.4 ergeben sich dabei folgende Forderungen:

- Die entstehende Relationenmatrix A soll möglichst dünn besetzt sein.
- Die zur Generierung der Relationen benötigte Zeit soll möglichst klein sein.

Die bei der Generierung der Relationen verbrauchte Zeit wird im wesentlichen beeinflußt durch

- die Zeit zur Berechnung des Potenzproduktes im Teilalgorithmus POTENZPROD,
- die Zeit zur Durchführung der Probedivisionen im Teilalgorithmus FAKTOR,
- die durchschnittliche Anzahl von Fehlversuchen, die auf einen Erfolg kommt.

Die durchschnittliche Anzahl der Fehlversuche wird bereits durch die Hinzunahme der ambigen Primformen zur Formenbasis deutlich herabgesetzt. Hier sind jedoch noch weitere Verbesserungen möglich.

Wir betrachten im nächsten Abschnitt zunächst die Wahl des Exponentenvektors $(x_1, \ldots, x_n)$. Durch geschickte Wahl dieses Vektors können wir die zur Berechnung des Potenzproduktes benötigte Zeit herabsetzen und die Struktur der entstehenden Matrix beeinflussen.

Im übernächsten Abschnitt wird die sogenannte “Early-Abort-Strategie” beschrieben, mit der die für die Probedivisionen benötigte Zeit verringert werden kann.

Im dann folgenden Abschnitt behandeln wir die sogenannte “Large-Prime-Variante”, mit deren Hilfe wir die Zahl der Fehlversuche weiter herabsetzen.

8.2 Wahl des Exponentenvektors

Wir erläutern in diesem Abschnitt unsere Vorgehensweise bei der Wahl der Exponentenvektoren. Ziel der Überlegungen ist es, die Einträge der Exponentenvektoren so zu wählen, daß zum einen die für die Berechnung des Potenzproduktes benötigte Zeit möglichst klein wird und zum anderen die entstehende Relationenmatrix dünn besetzt ist. Es wird sich herausstellen, daß beide Forderungen mit derselben Strategie zu erfüllen sind. Wir setzen dazu zunächst

$$k_0 = \max\{1 \le i \le k \ : \ p_i < \log D\}.$$

In der folgenden Tabelle wird deutlich, daß k_0 sehr klein ist im Vergleich zu n.

Tabelle 8.4 (Vergleich der Größe von n und k_0)

$\|\Delta\|$	n	k_0
$4 \cdot 10^{10} + 4$	249	7
$4 \cdot 10^{11} + 4$	300	5
$4 \cdot 10^{12} + 4$	339	5
$4 \cdot 10^{13} + 4$	391	5
$4 \cdot 10^{14} + 4$	458	7
$4 \cdot 10^{15} + 4$	483	5
$4 \cdot 10^{16} + 4$	556	9
$4 \cdot 10^{17} + 4$	613	6
$4 \cdot 10^{18} + 4$	649	7
$4 \cdot 10^{19} + 4$	708	6
$4 \cdot 10^{20} + 4$	796	9
$4 \cdot 10^{21} + 4$	896	7

$\|\Delta\|$	n	k_0
$4 \cdot 10^{22} + 4$	953	9
$4 \cdot 10^{23} + 4$	1001	7
$4 \cdot 10^{24} + 4$	1131	8
$4 \cdot 10^{25} + 4$	1181	8
$4 \cdot 10^{26} + 4$	1330	11
$4 \cdot 10^{27} + 4$	1304	8
$4 \cdot 10^{28} + 4$	1441	10
$4 \cdot 10^{29} + 4$	1511	11
$4 \cdot 10^{30} + 4$	1604	10
$4 \cdot 10^{31} + 4$	1754	11
$4 \cdot 10^{32} + 4$	1860	13
$4 \cdot 10^{33} + 4$	1933	6

$\|\Delta\|$	n	k_0
$4 \cdot 10^{34} + 4$	2010	17
$4 \cdot 10^{35} + 4$	2090	13
$4 \cdot 10^{36} + 4$	2238	10
$4 \cdot 10^{37} + 4$	2339	9
$4 \cdot 10^{38} + 4$	2439	13
$4 \cdot 10^{39} + 4$	2483	8
$4 \cdot 10^{40} + 4$	2695	15
$4 \cdot 10^{41} + 4$	2761	10
$4 \cdot 10^{42} + 4$	2939	8
$4 \cdot 10^{43} + 4$	3112	12
$4 \cdot 10^{44} + 4$	3216	16
$4 \cdot 10^{45} + 4$	3314	12

Die Grundidee unserer Strategie besteht nun darin, die Komponenten des Vektors x mit den Nummern $(k_0 + 1), \ldots, n$, also den weitaus größten Teil des Exponentenvektors, auf Null zu setzen. Nur die Komponenten $x_1, \ldots, x_{k_0}$ sollen mit Werten ungleich Null besetzt werden. Zur Berechnung des Potenzproduktes sind damit nur noch maximal k_0 Potenzen von Formen zu berechnen und miteinander zu multiplizieren.

Der Aufwand für die Berechnung der einzelnen Potenzen kann dabei ebenfalls herabgesetzt werden, wenn man die Einträge der nicht auf Null zu setzenden Komponenten sehr klein wählt. Wir legen also eine geeignete obere Schranke B fest, z.B. $B = 30$, und wählen

$$x_i \in [\![1, B]\!] \qquad (1 \le i \le k_0)$$

mit Hilfe eines Pseudo-Zufallsverfahrens.

Während der Generierung der Relationen werden nun die Potenzen der Formen $f_1, \ldots, f_{k_0}$ mit Exponenten zwischen 1 und B sehr häufig benötigt. Da k_0 und B klein sind, können diese Potenzen zu Beginn des Algorithmus RELATIONEN vorberechnet und gespeichert werden. Damit reduziert sich die Berechnung des Potenzproduktes auf die Multiplikation von k_0 vielen Formen.

Der Aufwand zur Berechnung eines einzelnen Potenzproduktes sinkt mit dieser Strategie im Vergleich zum Algorithmus von Hafner und McCurley auf einen Bruchteil. Für das in Kapitel 6 verwendete Beispiel ($|\Delta| = 10^{30}$) ergibt sich so eine Herabsetzung von über 250 000 Multiplikationen von Formenklassen pro Potenzprodukt auf etwa 10.

Aus diesen Überlegungen ergibt sich die im folgenden dargestellte erste Fassung des Teilalgorithmus POTENZPROD. Die Bezeichnung $1_{\mathcal{QF}}$ in Schritt (1) steht für die reduzierte Form in der Klasse $1_{\mathcal{Cl}}$.

Algorithmus 8.5 (Teilalgorithmus POTENZPROD (1. Fassung))

Eingabe: Δ, F und n wie oben angegeben sowie die Parameter k_0 und B, ferner die vorberechneten reduzierten Formen h_{ij} in den Klassen $[f_i^j]$ für $1 \leq i \leq k_0,\ 1 \leq j \leq B$

Ausgabe: ein Exponentenvektor $x \in \mathbb{Z}^n$ und die reduzierte Form $g \sim \prod_{i=1}^{n} f_i^{x_i}$

Setze $g \leftarrow 1_{\mathcal{QF}}$.		(1)
FOR $i = 1$ TO k_0		(2)
	Wähle x_i zufällig aus der Menge $[\![1, B]\!]$.	(3)
	Berechne die reduzierte Form g' in der Klasse $[g \cdot h_{i,x_i}]$ und setze $g \leftarrow g'$.	(4)
FOR $i = k_0 + 1$ TO n setze $x_i \leftarrow 0$.		(5)

Bei der Erprobung des Algorithmus hat es sich herausgestellt, daß noch eine weitere Beschleunigung bei der Berechnung des Potenzproduktes möglich ist. Wir besetzen dazu in Schritt (3) des Algorithmus POTENZPROD nicht alle Komponenten $x_1, \ldots, x_{k_0}$ mit zufällig gewählten Einträgen aus der Menge $[\![1, B]\!]$, sondern nur einen gewissen Anteil, z.B. die Hälfte. Die verbleibenden Komponenten des Exponentenvektors werden auf Null gesetzt. Die Teilmenge $T \subset [\![1, k_0]\!]$ mit den Nummern derjenigen Komponenten, die einen Eintrag ungleich Null erhalten, wird dabei für jeden Exponentenvektor neu zufällig zusammengestellt. Die Zahl der für die Berechnung eines Potenzproduktes notwendigen Multiplikationen von Formenklassen wird so nochmals herabgesetzt.

Auf diese Weise erhalten wir die im folgenden dargestellte zweite Fassung des Algorithmus POTENZPROD. Der Parameter t bezeichnet die Anzahl der mit Werten ungleich Null zu besetzenden Einträge im Exponentenvektor.

Algorithmus 8.6 (Teilalgorithmus POTENZPROD (2. Fassung))

Eingabe: Δ, F, n wie oben angegeben sowie die Parameter k_0, t und B, ferner die vorberechneten reduzierten Formen h_{ij} in den Klassen $[f_i^j]$ für $1 \le i \le k_0,\ 1 \le j \le B$

Ausgabe: ein Exponentenvektor $x \in \mathbb{Z}^n$ und die reduzierte Form $g \sim \prod_{i=1}^{n} f_i^{x_i}$

Wähle zufällig eine t-elementige Teilmenge T von $[\![1, k_0]\!]$.			(1)
Setze $g \leftarrow 1_{\mathcal{QF}}$.			(2)
FOR $i = 1$ TO k_0			(3)
	IF $i \in T$		(4)
	THEN	Wähle x_i zufällig aus der Menge $[\![1, B]\!]$.	(5)
		Berechne die reduzierte Form g' in der Klasse $[g \cdot h_{i,x_i}]$ und setze $g \leftarrow g'$.	(6)
	ELSE	Setze $x_i \leftarrow 0$.	(7)
FOR $i = k_0 + 1$ TO n setze $x_i \leftarrow 0$.			(8)

In der Praxis hat sich gezeigt, daß die hier beschriebene Vorgehensweise bei der Wahl der Exponentenvektoren keine negativen Auswirkungen auf die Erfolgswahrscheinlichkeit bei der Generierung der Relationen hat. Aus technischen Gründen muß t allerdings mindestens den Wert 4 haben.

Zur Initialisierung der Parameter für den Algorithmus POTENZPROD sind vor Schritt (1) des Algorithmus RELATIONEN die folgenden beiden Schritte einzufügen:

- Setze $k_0 \leftarrow \max\Big(\{1 \le i \le k \ : \ p_i < \log|\Delta|\} \cup \{4\}\Big)$, $t \leftarrow \max\{4, \lfloor \frac{k_0}{2} \rfloor\}$ und $B \leftarrow 30$.
- Berechne die reduzierten Formen h_{ij} in den Klassen $[f_i^j]$ für $1 \le i \le k_0$, $1 \le j \le B$. (Teilalgorithmus INITPOTENZ)

Der Algorithmus INITPOTENZ zur Vorberechnung der im Algorithmus POTENZPROD benötigten Potenzen hat dabei die folgende Gestalt:

Algorithmus 8.7 (Teilalgorithmus INITPOTENZ)

Eingabe: Δ und F wie oben angegeben sowie die Parameter k_0 und B

Ausgabe: die reduzierten Formen h_{ij} in den Klassen $[f_i^j]$ für $1 \le i \le k_0$, $1 \le j \le B$

FOR $i = 1$ TO k_0			(1)
	Setze $h_{i1} \leftarrow f_i$.		(2)
	FOR $l = 2$ TO B		(3)
		Berechne die reduzierte Form h_{il} in der Klasse $[h_{i,l-1} \cdot f_i]$.	(4)

Wir untersuchen nun die Auswirkungen unserer Vorgehensweise auf die Struktur der Relationenmatrix. Dazu müssen wir untersuchen, wie sich der endgültige Relationenvektor vom ursprünglichen Exponentenvektor unterscheidet. Der Relationenvektor entsteht aus dem Exponentenvektor, indem im Teilalgorithmus RELVEKTOR die einzelnen Einträge des Zerlegungsvektors zu den korrespondierenden Einträgen des Exponentenvektors addiert bzw. von diesen subtrahiert werden. Wir müssen also die Struktur des Zerlegungsvektors untersuchen, der im Teilalgorithmus FAKTOR ermittelt wird.

Zunächst ist festzustellen, daß der Koeffizient a der zu zerlegenden Form g nicht beliebig groß sein kann, da die Form g reduziert ist (vgl. (2.8)). Er zerfällt also sicher nicht in sehr viele Primfaktoren. Insofern haben die meisten Einträge des Zerlegungsvektors den Wert Null. Ferner kommen kleine Primfaktoren häufiger vor als große, d.h. insbesondere im unteren Teil des Zerlegungsvektors stehen viele Nullen. Schließlich treten bei großen Primteilern von a keine großen Exponenten auf, so daß die von Null verschiedenen Einträge im unteren Teil des Zerlegungsvektors sehr klein sind.

Aus dieser Überlegung ergibt sich, daß die Relationenvektoren eine ähnliche Struktur wie die Exponentenvektoren haben. Es sind allerdings auch im unteren Teil einige Einträge ungleich Null vorhanden, diese sind jedoch sehr klein, d.h. sie haben in der Regel den Wert ± 1, seltener den Wert ± 2, höhere Werte fast nie. Die dünn besetzte Struktur der Exponentenvektoren überträgt sich also im wesentlichen auf die Relationenvektoren. Es ergibt sich daher eine dünn besetzte Relationenmatrix mit den bereits in Abschnitt 6.4 angegebenen Eigenschaften.

8.3 Singularität der Relationenmatrix

Mit der im letzten Abschnitt beschriebenen Vorgehensweise für die Wahl der Exponentenvektoren haben wir das Ziel erreicht, zum einen die zur Berechnung der Potenzprodukte benötigte Zeit deutlich herabzusetzen und zum anderen eine dünn besetzte Relationenmatrix zu konstruieren. Allerdings ist die auf diese Weise konstruierte Matrix fast mit Sicherheit singulär, denn es ist sehr unwahrscheinlich, daß jede Primform aus der Formenbasis mindestens einmal als Faktor in einer erfolgreichen Faktorisierung auftritt. Diese Erwartung wird durch die praktische Erfahrung bestätigt. Insbesondere die Primformen mit großem p treten oft nicht auf. Da außerdem die meisten Komponenten der Exponentenvektoren immer den Wert Null haben, enthält die Relationenmatrix eine Reihe von Zeilen, die nur aus Nullen bestehen, und kann daher nicht regulär sein. Dies führt dazu, daß im weiteren Verlauf des Algorithmus CLASSGROUP eine große Anzahl zusätzlicher Relationen generiert werden muß, so daß das gesamte Verfahren ineffizient wird.

Die Generierung der Relationen muß also so gesteuert werden, daß die Relationenmatrix mit hoher Wahrscheinlichkeit regulär ist. Die Grundidee dazu ist es, im unteren Bereich der Exponentenvektoren jeweils an unterschiedlichen Positionen einen zusätzlichen Eintrag mit dem Wert 1 vorzusehen. Die Position dieses zusätzlichen Eintrages wird zufällig aus einer Menge S ausgewählt, die die Indizes derjenigen

Zeilen enthält, in denen noch mindestens ein Eintrag produziert werden muß, damit die Relationenmatrix mit hoher Wahrscheinlichkeit regulär ist. Dieser zusätzliche Eintrag überträgt sich, falls eine Relation gefunden wird, in den meisten Fällen auf den Relationenvektor und kann dann aus der Menge S entfernt werden.

Die Generierung der Relationen wird nun mindestens so lange fortgesetzt, bis die Menge S leer ist, auch wenn die entstehende Relationenmatrix dann geringfügig mehr als $(n+5)$ Spalten hat. In der Praxis ist diese Fortsetzung jedoch fast nie erforderlich. Es hat sich gezeigt, daß der beschriebene Steuerungsmechanismus in der Regel dafür sorgt, daß die Menge S leer ist, bevor $(n+5)$ Relationen generiert worden sind.

Der Rahmenalgorithmus RELATIONEN erhält damit die folgende Form:

Algorithmus 8.8 (Rahmenalgorithmus RELATIONEN (2. Fassung))

Eingabe: Δ, F, F_p, n, k und M wie oben angegeben

Ausgabe: eine Matrix $A \in \mathbb{Z}^{n \times m}$ mit $m \geq n+5$, deren Spalten Relationen über F sind

Setze $k_0 \leftarrow \max\Big(\{1 \leq i \leq k_1 \ : \ p_i < \log|\Delta|\} \cup \{4\}\Big)$, $t \leftarrow \max\{4, \lfloor \frac{k_0}{2} \rfloor\}$. Setze $B \leftarrow 30$. (1)

Berechne die reduzierten Formen h_{ij} in den Klassen $[f_i^j]$ für $1 \leq i \leq k_0$, $1 \leq j \leq B$. (Teilalgorithmus INITPOTENZ) (2)

Setze $S \leftarrow \{1, \ldots, n\}$ und $m \leftarrow 1$. (3)

WHILE $m \leq n+5$ oder $S \neq \emptyset$ (4)

  Wähle $s \in S$ zufällig. (5)

  Wähle einen Exponentenvektor $x = (x_1, \ldots, x_n) \in \mathbb{Z}^n$ mit $x_s \neq 0$ und berechne die reduzierte Form $g = (a, b)$ in der Klasse von $\prod_{i=1}^{n} f_i^{x_i}$. (Teilalgorithmus POTENZPROD) (6)

  Stelle fest, ob a vollständig über F_p zerfällt, und, falls ja, berechne den Zerlegungsvektor $y = (y_1, \ldots, y_n) \in \mathbb{Z}^n$ mit $a = \prod_{i=1}^{n} p_i^{y_i}$. Gib ferner den unzerlegbaren Anteil r von a an. (Teilalgorithmus FAKTOR) (7)

  UNTIL $r = 1$. (8)

  Berechne den Relationenvektor $w = (w_1, \ldots, w_n) \in \mathbb{Z}^n$. (Teilalgorithmus RELVEKTOR) (9)

  IF w ist nicht der Nullvektor (10)

  THEN

    IF $S' = \{\nu \in S \ : w_\nu \neq 0\} \neq \emptyset$ THEN entferne $\max S'$ aus S. (11)

    Setze (Spalte) $A_m \leftarrow w$ und $m \leftarrow m+1$. (12)

Der Algorithmus POTENZPROD muß ebenfalls um einige Schritte erweitert werden:

Algorithmus 8.9 (Teilalgorithmus POTENZPROD (Endfassung))

Eingabe: Δ, F, n wie oben angegeben sowie die Parameter k_0, t, B und s, ferner die vorberechneten reduzierten Formen h_{ij} in den Klassen $[f_i^j]$ für $1 \leq i \leq k_0,\ 1 \leq j \leq B$

Ausgabe: ein Exponentenvektor $x \in \mathbb{Z}^n$ mit $x_s \neq 0$ und die reduzierte Form $g \sim \prod_{i=1}^{n} f_i^{x_i}$

Wähle zufällig eine t-elementige Teilmenge T von $[\![1, k_0]\!]$.				(1)
Setze $g \leftarrow 1_{\mathcal{QF}}$.				(2)
FOR $i = 1$ TO k_0				(3)
	IF $i \in T$			(4)
	THEN	Wähle x_i zufällig aus der Menge $[\![1, B]\!]$.		(5)
		Berechne die reduzierte Form g' in der Klasse $[g \cdot h_{i,x_i}]$ und setze $g \leftarrow g'$.		(6)
	ELSE	Setze $x_i \leftarrow 0$.		(7)
FOR $i = k_0 + 1$ TO n setze $x_i \leftarrow 0$.				(8)
IF $s \notin T$				(9)
THEN	Setze $x_s \leftarrow 1$.			(10)
	Berechne die reduzierte Form g' in der Klasse $[g \cdot h_{i,1}]$ und setze $g \leftarrow g'$.			(11)

Man beachte, daß die hier beschriebene Vorgehensweise keine Garantie für die Regularität der Relationenmatrix bietet, es wird lediglich die Wahrscheinlichkeit dafür erhöht. Da es keinen schnellen Test gibt, der mit Sicherheit die Regularität einer Matrix feststellt, muß man die verbleibende Unsicherheit in Kauf nehmen, daß sich die Relationenmatrix später als singulär erweisen kann. Die einzige Möglichkeit, diesen Fall mit Sicherheit auszuschließen, wäre die Generierung einer diagonaldominanten Matrix, was aber aus den in Abschnitt 6.1 genannten Gründen in der Praxis viel zu aufwendig ist. In der Praxis zeigt sich zudem, daß die mit der letzten Version des Algorithmus RELATIONEN generierte Relationenmatrix in fast allen Fällen tatsächlich regulär ist.

8.4 Die Early-Abort-Strategie

Wir untersuchen nun, wie der Faktorisierungsprozeß im Teilalgorithmus FAKTOR beschleunigt werden kann. Dazu sei a der zu zerlegende erste Koeffizient der zuvor mit dem Teilalgorithmus POTENZPROD berechneten reduzierten Form. Die grundsätzliche Vorgehensweise des Algorithmus FAKTOR ist es, mit Hilfe von Probedivisionen eine vollständige Zerlegung von a über F_p zu finden.

Für die Untersuchung des Faktorisierungsprozesses ist es von Bedeutung, daß bei steigender Diskriminante die Erfolgswahrscheinlichkeit bei der Generierung der Relationen sinkt. Für $|\Delta| \approx 10^{30}$ liegt die Erfolgswahrscheinlichkeit etwa bei $1/100$, für $|\Delta| \approx 10^{40}$ etwa bei $1/1000$. Im letzteren Fall sind also im Schnitt mehr als 1000 Versuche nötig, um eine Relation zu finden. Die weitaus meisten Faktorisierungsversuche sind also nicht erfolgreich.

Man fragt sich daher, ob man bereits vor der Durchführung aller Probedivisionen mit relativ hoher Sicherheit vorhersagen kann, ob a über F_p vollständig zerfällt oder nicht. Ist dies möglich, so wird man den Zerlegungsversuch nur dann fortsetzen, wenn man relativ sicher ist, daß a tatsächlich vollständig über F_p zerfällt. Andernfalls wird man die Faktorisierung sofort abbrechen. Dabei muß man das Risiko in Kauf nehmen, in einigen wenigen Fällen auch die Zerlegung einer Zahl a abzubrechen, die doch vollständig über F_p zerfällt. Andererseits gewinnt man jedoch die Rechenzeit, die für die vollständige Probedivision vieler Zahlen a, die nicht zerfallen, verbraucht würde.

Eine solche Vorgehensweise wird im allgemeinen als **Early-Abort-Strategie** bezeichnet. Diese Bezeichnung weist darauf hin, daß man relativ früh während des Prozesses der Probedivisionen entscheidet, ob eine Zahl ausscheidet oder nicht. Die Idee der Early-Abort-Strategie ist von einigen Faktorisierungsalgorithmen und auch im Zusammenhang mit dem Index-Calculus-Verfahren zur Berechnung diskreter Logarithmen (vgl. z.B. [32], [33]) bekannt.

Wir übertragen im folgenden diese Idee auf unsere Anwendung. Wir legen dazu zunächst zwei Parameter $k_1, k_2 \in [\![1, n]\!]$ mit $k_1 < k_2$ geeignet fest, z.B. $k_1 = \lfloor \frac{n}{4} \rfloor$ und $k_2 = \lfloor \frac{n}{2} \rfloor$. Dabei soll k_2 mindestens so groß gewählt werden, daß $p_{k_2}^2 > p_k$ ist.

Die Schritte (1) bis (5) der bisherigen Version des Algorithmus FAKTOR bleiben unverändert. Der anschließende Vorgang der Probedivisionen durch die Primzahlen $p_1, \ldots, p_k$ wird nun in drei Abschnitte eingeteilt.

Zunächst wird die vollständige Probedivision durch alle Primzahlen $p_1, \ldots, p_{k_1}$ durchgeführt. Danach wird der noch nicht zerlegte Anteil r mit 2^{32} verglichen.

Ist $r \geq 2^{32}$, so zerfällt r mit hoher Wahrscheinlichkeit nicht über den verbleibenden Primzahlen aus F_p. Andernfalls müßte a sehr viele große Primfaktoren enthalten, was sehr unwahrscheinlich ist. Der Zerlegungsprozeß wird daher in diesem Fall vorzeitig abgebrochen.

Ist aber $r < 2^{32}$, so wird die Faktorisierung mit der vollständigen Probedivision durch die Primzahlen $p_{k_1+1}, \ldots, p_{k_2}$ fortgesetzt. Diese Probedivisionen können nun, da $r < 2^{32}$ ist, in der Praxis wesentlich schneller ausgeführt werden als Probedivisionen größerer Zahlen.

Anschließend wird r mit p_k verglichen. Falls nun $r > p_k$ ist, so wird die Faktorisierung aus demselben Grund wie oben vorzeitig abgebrochen. Ist jedoch $r \leq p_k$, so können wir mit Hilfe des folgenden Satzes bereits an dieser Stelle schließen, daß eine vollständige Zerlegung von a über F_p möglich ist.

Satz 8.10 *Es sei $P \subseteq I\!P$ und c eine natürliche Zahl, die sich als Potenzprodukt der Primzahlen aus P darstellen läßt. Es sei ferner $d \in I\!N$, so daß $p \nmid c$ für alle Primzahlen $p \in P$ mit $p \leq d$. Dann gilt: Ist $c < d^2$, so ist $c \in P$.*

Beweis: Wir nehmen an, c sei nicht in P. Dann existieren $p_1, p_2 \in P$, so daß $p_1 p_2 \mid c$. Nach Voraussetzung gilt $p_1 > d$ und $p_2 > D$. Daraus folgt $c > d^2$, was der an c gestellten Voraussetzung widerspricht. ■

In unserer Situation ist $P = \{p \in I\!P \; : \; (\frac{\Delta}{p}) = 1\}$, $d = p_{k_2}$ und $c = r$. Offensichtlich zerfällt r vollständig über P. Da ferner $r \leq p_k$ ist und k_2 so gewählt war, daß $p_{k_2}^2 > p_k$ ist, folgt $r < p_{k_2}^2$, es ist also nach obigem Satz $r \in P$. Da F_p alle Primzahlen $p \leq p_k$ enthält, die in P liegen, ist $r \in F_p$.

Da nun also nicht nur feststeht, daß r über F_p zerlegbar ist, sondern wir darüberhinaus sogar wissen, daß $r \in F_p$ ist, sind nur noch Vergleiche nötig, um die Faktorisierung von a zu vollenden. Die Primzahlen $p_{k_2+1}, \ldots, p_k$ treten also bei dieser Vorgehensweise niemals in Probedivisionen auf.

Insgesamt ergibt sich aus diesen Überlegungen die folgende endgültige Version des Teilalgorithmus FAKTOR:

Algorithmus 8.11 (Teilalgorithmus FAKTOR (Endfassung))

Eingabe: der Koeffizient $a \in \mathbb{N}$ einer reduzierten Form aus $\mathcal{QF}(\Delta)$ sowie F_p, n, k und M wie oben angegeben, ferner die Parameter k_1 und k_2

Ausgabe: der zu a gehörende Zerlegungsvektor $y = (y_1, \ldots, y_n) \in \mathbb{Z}^n$ sowie r wie in Abschnitt 8.1 angegeben

IF a wird von einer der Primzahlen in der Menge M geteilt THEN setze $r \leftarrow 0$ und EXIT.			(1)
Setze $r \leftarrow a$.			(2)
FOR $i = 1$ TO n setze $y_i \leftarrow 0$.			(3)
FOR $i = k_1 + 1$ TO n			(4)
	IF $p_i \mid r$ THEN setze $r \leftarrow r/p_i$ und setze $y_i \leftarrow 1$.		(5)
Setze $i \leftarrow 1$.			(6)
WHILE $i \leq k_1$ und $r \geq 2^{32}$			(7)
	WHILE $p_i \mid r$ (großes r: aufwendige Probedivisionen)		(8)
		Setze $r \leftarrow r/p_i$ und erhöhe y_i um 1.	(9)
	IF $r = 1$ THEN EXIT.		(10)
	Erhöhe i um 1.		(11)
IF $r \geq 2^{32}$ THEN setze $r \leftarrow 0$ und EXIT.			(12)
WHILE $i \leq k_2$			(13)
	WHILE $p_i \mid r$ (kleines r: schnelle Probedivisionen)		(14)
		Setze $r \leftarrow r/p_i$ und erhöhe y_i um 1.	(15)
	IF $r = 1$ THEN EXIT.		(16)
	Erhöhe i um 1.		(17)
IF $r > p_{k_2}^2$ THEN setze $r \leftarrow 0$ und EXIT.			(18)
IF $r > p_k$ THEN EXIT.			(19)
WHILE $r \neq p_i$ (nur noch Vergleiche)			(20)
	Erhöhe i um 1.		(21)
Setze $y_i \leftarrow 1$ und $r \leftarrow 1$.			(22)

Vor dem ersten Aufruf des Algorithmus FAKTOR müssen im Rahmenalgorithmus RELATIONEN die Parameter k_1 und k_2 geeignet gewählt werden.

Der Grund für die Wahl der Konstanten 2^{32} in den Schritten (7) bzw. (12) ist ausschließlich die bereits erwähnte Tatsache, daß für $r < 2^{32}$ die Probedivisionen in der Schleife (13) - (17) in der Praxis wesentlich schneller ausgeführt werden können als für größere Zahlen.

Wenn nach Abschluß der Schleife (13) - (17) des Algorithmus FAKTOR

$$p_k < r \leq p_{k_2}^2$$

gilt, so ist zwar a nicht über F_p zerlegbar, mit Satz 8.10 ergibt sich jedoch, daß der bisher noch nicht zerlegte Anteil r von a eine Primzahl ist. In diesem Fall ist also r der unzerlegbare Anteil von a wie in Abschnitt 8.1 angegeben. Ist jedoch an dieser Stelle

$$r > p_{k_2}^2,$$

so kann man nicht schließen, daß r prim ist. In diesem Fall kann r sogar noch Primfaktoren in $\{p_{k_2+1}, \ldots, p_k\}$ haben. Es steht also nicht fest, ob r der über F_p unzerlegbare Anteil von a ist. Aus diesem Grund wird in Schritt (18) vor dem Verlassen des Algorithmus FAKTOR $r = 0$ gesetzt.

Die Verwendung der Early-Abort-Strategie führt in der Praxis zu einer wesentlichen Beschleunigung bei der Generierung der Relationen. Für $|\Delta| \approx 10^{30}$ sinkt die Summe der Rechenzeiten aus den einzelnen Aufrufen des Algorithmus FAKTOR um etwa 50%. Die insgesamt zur Generierung der Relationen benötigte Zeit wird dadurch um etwa 35% herabgesetzt. Für größere Diskriminanten steigt der Zeitgewinn noch mehr an.

8.5 Die Large-Prime-Variante

Wir untersuchen nun die Frage, wie die durchschnittliche Anzahl der Fehlversuche zur Generierung einer Relation weiter herabgesetzt werden kann. Dazu übertragen wir die Idee der **Large-Prime-Variante**, die wie die Early-Abort-Strategie von Faktorisierungsalgorithmen und von der Berechnung diskreter Logarithmen her bekannt ist (vgl. z.B. [33], [32]), auf unsere Situation.

Zur Generierung einer Relation wird im Teilalgorithmus POTENZPROD zunächst ein Exponentenvektor $x \in \mathbb{Z}^n$ gewählt und dann die reduzierte Form g berechnet, die der Bedingung

$$g \sim \prod_{i=1}^{n} f_i^{x_i} \tag{8.1}$$

genügt. Anschließend versucht man, eine vollständige Zerlegung dieser Form g über der Formenbasis F zu finden, d.h. eine Darstellung

$$g \sim \prod_{i=1}^{n} f_i^{y_i}. \tag{8.2}$$

Wenn dies nicht gelingt, kann mit dem Exponentenvektor x keine Relation gefunden werden.

Wir betrachten nun auch unvollständige Zerlegungen der Form g, also Zerlegungen der Form

$$g \;\sim\; h^{e_h} \cdot \prod_{i=1}^{n} f_i^{y_i}, \tag{8.3}$$

wobei $h = (p_h, b_h)$ eine Primform in $\mathcal{QF}(\Delta)$ und $e_h = \pm 1$ ist. Aus dieser unvollständigen Zerlegung kann man mit Gleichung (8.1) eine sogenannte **unvollständige Relation**

$$h \sim \prod_{i=1}^{n} f_i^{e_h(x_i - y_i)} \tag{8.4}$$

konstruieren. Wir bezeichnen die Form h als die **Zerlegungsrestform** der unvollständigen Relation (8.4).

Tritt nun die Primform h in einer weiteren unvollständigen Relation auf, so kann man aus den beiden unvollständigen Relationen eine vollständige Relation zusammensetzen. Sind nämlich für $j = 1, 2$

$$h_j \sim \prod_{i=1}^{n} f_i^{e_j(x_{ij} - y_{ij})} \tag{8.5}$$

die beiden unvollständigen Relationen, so ist

$$\begin{aligned} 1_{\mathcal{QF}} \ \sim \ h \cdot h^{-1} \ &\sim \ \prod_{i=1}^{n} f_i^{e_1(x_{i1} - y_{i1}) - e_2(x_{i2} - y_{i2})} && (8.6) \\ &\sim \ \prod_{i=1}^{n} f_i^{(x_{i1} - y_{i1}) - e_1 e_2(x_{i2} - y_{i2})} && (8.7) \end{aligned}$$

eine vollständige Relation.

Wir nutzen diesen Zusammenhang aus, indem wir unvollständige Relationen speichern und hoffen, daß dieselbe Primform später noch einmal als Zerlegungsrestform auftritt. Nun sinkt mit steigender Größe von p_h die Wahrscheinlichkeit, daß eine Form h nochmals als Zerlegungsrestform auftritt. Daher wählen wir eine geeignete Konstante L und betrachten nur unvollständige Zerlegungen mit $p_h \leq L$.

Aus Satz 7.12 folgt, daß eine solche unvollständige Zerlegung der Form $g = (a, b)$ genau dann existiert, wenn bei der Faktorisierung von a über der Menge F_p der zu zu den Primformen aus F gehörenden Primzahlen ein Rest $r < L$ bleibt, der nachweislich prim ist. Nun gibt der Algorithmus FAKTOR, wie am Ende des vorigen Abschnittes erläutert wurde, für r nur dann einen Wert größer als 1 aus, wenn dieser prim ist. Die Primalität von r muß daher hier nicht mehr gesondert untersucht werden.

Im Zusammenhang mit der Large-Prime-Variante ist ferner die Tatsache von Bedeutung, daß der Wert des zweiten Koeffizienten b_h der Primform $h = (p_h, b_h)$ nicht explizit berechnet werden muß. Aus Gleichung (8.7) ist ersichtlich, daß die explizite Kenntnis der Vorzeichen der beiden Exponenten e_1 und e_2 zur Konstruktion der vollständigen Relation nicht erforderlich ist. Es reicht vielmehr aus, das Vorzeichen von $e = -e_1 \cdot e_2$ zu kennen, also zu wissen, ob die Vorzeichen der Exponenten der Primform h in den beiden unvollständigen Relationen übereinstimmen oder nicht. Dies kann man feststellen, ohne beide Koeffizienten der Form h zu kennen, denn aus Satz 7.12 folgt unmittelbar:

Satz 8.12 *Es sei $\Delta \in \mathcal{D}^-$. Es seien $g_1 = (a_1, b_1)$ und $g_2 = (a_2, b_2)$ Primformen in $\mathcal{QF}(\Delta)$, die den Voraussetzungen des Satzes 7.12 genügen. Es sei $h = (p_h, b_h)$ eine (gewöhnliche oder ambige) Primform in $\mathcal{QF}(\Delta)$, so daß p_h sowohl a_1 als auch a_2 genau zur ersten Potenz teilt. Es seien e_1 und e_2 die Exponenten von h in der Zerlegung von g_1 und g_2 in Primformen gemäß Satz 7.12. Dann ist*

$$b_1 \equiv (e_1 e_2) \cdot b_2 \; mod \, (2p_h).$$

Für das Zusammensetzen zweier unvollständiger Relationen reicht es also aus, die beiden zweiten Koeffizienten b_1 und b_2 der Formen g_1 und g_2 zu kennen, bei deren Faktorisierung die unvollständigen Relationen entstanden sind. Wir speichern daher die einzelnen unvollständigen Relationen in der Form (p_h, w, b), wobei

- $w \in \mathbb{Z}^n$ mit $w_i = x_i - y_i$ $(1 \leq i \leq n)$ der unvollständige Relationenvektor gemäß Gleichung (8.4) ist,
- p_h der erste Koeffizient der Zerlegungsrestform h gemäß Gleichung (8.4) ist,
- b der zweite Koeffizient der Form g aus Gleichung (8.3) ist, bei deren Zerlegung die unvollständige Relation entstanden ist.

Zur Konstruktion einer vollständigen Relation v aus zwei unvollständigen Relationen (p, w, b) und (p', w', b') mit $p' = p$ ergibt sich dann der folgende Algorithmus:

Algorithmus 8.13 (Teilalgorithmus LARGEREL)

Eingabe: zwei unvollständige Relationen (p, w, b) und (p', w', b') mit $p' = p$ sowie n und k wie oben angegeben

Ausgabe: eine vollständige Relation $v \in \mathbb{Z}^n$

IF $b_1 \equiv b_2$ mod $(2p_h)$		(1)
THEN	FOR $i = 1$ TO n setze $v_i \leftarrow w_i - w'_i$.	(2)
ELSE	FOR $i = 1$ TO n setze $v_i \leftarrow w_i + w'_i$.	(3)
FOR $i = k + 1$ TO n reduziere v_i modulo 2.		(4)

In Schritt (4) werden die zu den ambigen Primformen gehörenden Einträge des neuen Relationenvektors modulo 2 reduziert. Dies ist erlaubt, da die entsprechenden Äquivalenzklassen die Ordnung 2 haben (vgl. Satz 7.10).

Der Rahmenalgorithmus RELATIONEN muß für die Verwendung der Large-Prime-Variante wie folgt ergänzt werden:

Algorithmus 8.14 (Rahmenalgorithmus RELATIONEN (Endfassung))

Eingabe: Δ, F, F_p, n, k und M wie oben angegeben
Ausgabe: eine Matrix $A \in \mathbb{Z}^{n\times m}$ mit $m \geq n+5$, deren Spalten Relationen über F sind

Setze $k_0 \leftarrow \max\left(\{1 \leq i \leq k_1 \ : \ p_i < \log\lvert\Delta\rvert\} \cup \{4\}\right)$, $t \leftarrow \max\{4, \lfloor \frac{k_0}{2} \rfloor\}$. Setze $B \leftarrow 30$.			(1)
Berechne die reduzierten Formen h_{ij} in den Klassen $[f_i^j]$ für $1 \leq i \leq k_0$, $1 \leq j \leq B$. (Teilalgorithmus INITPOTENZ)			(2)
Setze $k_1 \leftarrow \lfloor \frac{n}{4} \rfloor$. Setze $k_2 \leftarrow \max\{\lfloor \frac{n}{2} \rfloor, n'\}$, wobei $n' = \min\{1 \leq i \leq k \ : p_i^2 > p_k\}$ ist.			(3)
Setze $S \leftarrow \{1, \ldots, n\}$, $m \leftarrow 1$ und $L \leftarrow p_{k_2}^2$.			(4)
WHILE $m \leq n+5$ oder $S \neq \emptyset$			(5)
	Wähle $s \in S$ zufällig.		(6)
	Wähle einen Exponentenvektor $x = (x_1, \ldots, x_n) \in \mathbb{Z}^n$ mit $x_s \neq 0$ und berechne die reduzierte Form $g = (a, b)$ in der Klasse von $\prod_{i=1}^{n} f_i^{x_i}$. (Teilalgorithmus POTENZPROD)		(7)
	Stelle fest, ob a vollständig über F_p zerfällt und, falls ja, berechne den Zerlegungsvektor $y = (y_1, \ldots, y_n) \in \mathbb{Z}^n$ mit $a = \prod_{i=1}^{n} p_i^{y_i}$. Gib ferner den unzerlegbaren Anteil r von a an. (Teilalgorithmus FAKTOR)		(8)
	UNTIL $r \leq L$.		(9)
	Berechne den (vollständigen oder unvollständigen) Relationenvektor $w = (w_1, \ldots, w_n) \in \mathbb{Z}^n$. (Teilalgorithmus RELVEKTOR)		(10)
	IF $r > 1$		(11)
	THEN	IF in der Liste der unvollständigen Relationen existiert ein Eintrag (r', w', b') mit $r' = r$	(12)
		THEN: Berechne aus (r, w, b) und (r', w', b') eine vollständige Relation $v \in \mathbb{Z}^n$. (Teilalgorithmus LARGEREL)	(13)
		Setze $w \leftarrow v$ und $r \leftarrow 1$.	(14)
		ELSE: Speichere (r, w, b) in der Liste der unvollständigen Relationen.	(15)
	IF $r = 1$ und w ist nicht der Nullvektor		(16)
	THEN	IF $S' = \{\nu \in S \ : w_\nu \neq 0\} \neq \emptyset$ THEN entferne $\max S'$ aus S.	(17)
		Setze (Spalte) $A_m \leftarrow w$ und $m \leftarrow m+1$.	(18)

Die Tabelle auf der folgenden Seite zeigt den Effekt der Large-Prime-Variante und gibt Auskunft über die Erfolgswahrscheinlichkeit des Verfahrens zur Generierung der Relationen. Sie enthält im einzelnen die folgenden Daten:

- die absolute Diskriminante $|\Delta|$,
- die Anzahl m der insgesamt generierten Relationen,
- die darin enthaltene Anzahl m_l der mit Hilfe der Large-Prime-Variante zustandegekommenen Relationen,
- den prozentualen Anteil t_l der mit Hilfe der Large-Prime-Variante zustandegekommenen Relationen an der Gesamtzahl der Relationen,
- die Anzahl l der im Verlauf des Verfahrens gespeicherten unvollständigen Relationen,
- die Anzahl v der Versuche (Wahl eines Exponentenvektors, Bildung des Potenzproduktes, Versuch der Zerlegung), die zur Generierung aller Relationen erforderlich waren,
- die durchschnittliche Anzahl $\overline{v}$ von Versuchen, die zur Generierung einer Relation erforderlich waren.

Es wird deutlich, daß der Erfolg der Large-Prime-Variante mit steigender Diskriminante größer wird. Für Diskriminanten mit mehr als 25 Dezimalstellen werden in der Regel etwa 12 - 16% der Relationen mit Hilfe der Large-Prime-Variante generiert. Dies führt in der Praxis zu einer deutlichen Herabsetzung der Gesamtlaufzeit.

Für sehr kleine Diskriminanten wird nur ein sehr geringer Anteil der Relationen mit Hilfe der Large-Prime-Variante gefunden. Hier ist in der Praxis wegen des trotzdem vorhandenen Aufwandes zur Verwaltung der unvollständigen Relationen keine Verbesserung der Gesamtlaufzeit zu verzeichnen.

Aus den letzten beiden Spalten der Tabelle läßt sich ferner entnehmen, daß die Erfolgswahrscheinlichkeit des Verfahrens zur Generierung der Relationen für große Diskriminanten stark abnimmt. Hier werden im Durchschnitt immer mehr Versuche benötigt, um eine Relation zu generieren.

Die Ergebnisse für $|\Delta| = 4 \cdot 10^{33} + 4$ und $|\Delta| = 4 \cdot 10^{39} + 4$ fallen auf, da hier die Erfolgswahrscheinlichkeit wesentlich geringer ist als erwartet. Der Grund hierfür ist die Tatsache, daß bei diesen beiden Diskriminanten jeweils ein mehrfacher Primteiler der Diskriminante vorhanden ist ($p = 11$ bzw. $p = 13$), der nach den Ausführungen in Abschnitt 7.2 in der Formenbasis nicht durch eine Primform repräsentiert ist. Weitere mehrfache Primteiler der Diskriminante treten bei $|\Delta| = 4 \cdot 10^{11} + 4$ ($p = 11$) und bei $|\Delta| = 4 \cdot 10^{21} + 4$ ($p = 7$) auf. Bei diesen kleineren Diskriminanten macht sich die Herabsetzung der Erfolgswahrscheinlichkeit jedoch nicht so stark bemerkbar.

Tabelle 8.15 (Effekt der Large-Prime-Variante)

$\|\Delta\|$	m	m_l	t_l	l	v	$\overline{v}$
$4 \cdot 10^{10} + 4$	254	1	0.4	51	319	1.3
$4 \cdot 10^{11} + 4$	305	5	1.6	133	537	1.8
$4 \cdot 10^{12} + 4$	344	6	1.7	222	764	2.2
$4 \cdot 10^{13} + 4$	390	13	3.3	261	924	2.4
$4 \cdot 10^{14} + 4$	464	11	2.4	235	865	1.9
$4 \cdot 10^{15} + 4$	488	24	4.9	455	1601	3.3
$4 \cdot 10^{16} + 4$	561	27	4.8	396	1356	2.4
$4 \cdot 10^{17} + 4$	618	41	6.6	686	2444	3.9
$4 \cdot 10^{18} + 4$	654	54	8.3	918	3448	5.3
$4 \cdot 10^{19} + 4$	713	58	8.1	1011	4044	5.7
$4 \cdot 10^{20} + 4$	801	45	5.6	1099	4470	5.6
$4 \cdot 10^{21} + 4$	901	72	8.0	1366	7162	7.9
$4 \cdot 10^{22} + 4$	958	77	8.0	1455	7108	7.4
$4 \cdot 10^{23} + 4$	1006	116	11.5	1842	11785	11.7
$4 \cdot 10^{24} + 4$	1136	132	11.6	1955	13540	11.9
$4 \cdot 10^{25} + 4$	1186	157	13.2	2393	22141	18.7
$4 \cdot 10^{26} + 4$	1335	139	10.4	2345	18729	14.0
$4 \cdot 10^{27} + 4$	1309	153	11.7	2746	30965	23.7
$4 \cdot 10^{28} + 4$	1446	194	13.4	2994	33710	23.3
$4 \cdot 10^{29} + 4$	1516	190	12.5	3269	47690	31.5
$4 \cdot 10^{30} + 4$	1609	220	13.7	3582	63169	39.3
$4 \cdot 10^{31} + 4$	1759	256	14.6	3758	65970	37.5
$4 \cdot 10^{32} + 4$	1865	259	13.9	4120	86723	46.5
$4 \cdot 10^{33} + 4$	1938	427	22.0	4536	253164	130.6
$4 \cdot 10^{34} + 4$	2015	288	14.3	4640	134402	66.7
$4 \cdot 10^{35} + 4$	2095	328	15.7	5266	216898	103.5
$4 \cdot 10^{36} + 4$	2243	333	14.8	5495	357768	159.5
$4 \cdot 10^{37} + 4$	2344	392	16.4	6310	739291	315.4
$4 \cdot 10^{38} + 4$	2444	345	14.1	6439	489310	200.2
$4 \cdot 10^{39} + 4$	2488	454	18.2	6976	1484951	596.9
$4 \cdot 10^{40} + 4$	2700	368	13.6	7108	707489	262.0
$4 \cdot 10^{41} + 4$	2766	422	15.3	7983	1829995	661.6
$4 \cdot 10^{42} + 4$	2944	464	15.8	8340	2410622	818.8
$4 \cdot 10^{43} + 4$	3117	419	13.4	8878	2897106	929.5
$4 \cdot 10^{44} + 4$	3221	424	13.2	9144	2516956	781.4
$4 \cdot 10^{45} + 4$	3319	402	12.1	9713	4925298	1484.0

Kapitel 9

Die Verkleinerung der Relationenmatrix

In diesem Kapitel beschreiben wir Schritt (4) des Algorithmus CLASSGROUP. Hier wird die zuvor in Schritt (3) generierte Relationenmatrix A unter Ausnutzung ihrer dünn besetzten Struktur möglichst stark verkleinert. Aus dem Relationensystem $\mathcal{R} = (F, A, n, m)$ wird so das Relationensystem $\mathcal{R}_1 = (F_1, B, n_1, m_1)$ mit $n_1 < n$, $F_1 \subset F$ und $\operatorname{Det} \Lambda(A) = \operatorname{Det} \Lambda(B)$ konstruiert.

Auch der Algorithmus zur Verkleinerung der Relationenmatrix wird in einen Rahmenalgorithmus und einige Teilalgorithmen unterteilt. Für den Rahmenalgorithmus werden wie in Kapitel 8 mehrere Versionen angegeben.

9.1 Die Grundidee des Verfahrens

Das Verfahren zur Verkleinerung der Relationenmatrix A wurde bereits in Abschnitt 6.6 motiviert. Wir geben die Grundidee hier zunächst nochmals an.

Dazu sei $A \in \mathbb{Z}^{n\times m}$ die vorgegebene Relationenmatrix und $F = \{f_1, \ldots, f_n\}$ die entsprechende Formenbasis. Existieren nun Indizes $i \in [\![1, n]\!]$ und $j \in [\![1, m]\!]$, so daß

$$\begin{aligned} a_{ij} &= \pm 1, \\ a_{ik} &= 0 \qquad (1 \leq k \leq m,\ k \neq j) \end{aligned} \tag{9.1}$$

ist, so gilt

$$f_i \sim \prod_{l=1}^{i-1} f_l^{\pm a_{lj}} \cdot \prod_{l=i+1}^{n} f_l^{\pm a_{lj}},$$

wobei das positive Vorzeichen der Exponenten a_{lj} genau dann gilt, wenn $a_{ij} = 1$ ist. Streicht man nun die i-te Zeile und die j-te Spalte aus der Matrix A, so sind nach Satz 6.6 die Spalten der entstehenden Matrix $A' \in \mathbb{Z}^{(n-1)\times(m-1)}$ Relationen über der Formenbasis $F' = F \setminus \{f_i\}$. Die Grundidee des Verfahrens besteht also darin, sukzessive geeignete Zeilen und Spalten der Relationenmatrix zu streichen und gleichzeitig die Formenbasis entsprechend zu verkleinern.

Wir bezeichnen im folgenden die Bedingung (9.1) als die **Eliminationsbedingung** (für Zeile i und Spalte j), den Vorgang des Streichens einer Zeile und Spalte als **Elimination**.

Wir wählen nun in jedem Schritt des Verfahrens eine solche Zeile zur Elimination aus, in der die Eliminationsbedingung erfüllt ist bzw. leicht erfüllt werden kann. Letzteres ist der Fall, wenn zur Erfüllung der Eliminationsbedingung nur wenige Spaltenoperationen erforderlich sind, die zudem nur geringe Auswirkungen auf die Besetzung der Matrix oder die Größe der Matrixeinträge haben. Dazu wiederum ist es notwendig, daß die zu eliminierende Zeile sehr viele Nullen enthält und daß alle von Null verschiedenen Einträge in dieser Zeile den Wert ± 1 haben.

Enthält nämlich eine zu eliminierende Zeile bereits viele Nullen, so sind nur wenige Spaltenoperationen erforderlich, um die restlichen Einträge dieser Zeile zu Null zu machen. Dies führt dazu, daß nur wenige Einträge in anderen Zeilen, die vorher Null waren, durch Werte ungleich Null ersetzt werden. Die dünne Besetzung der Matrix wird auf diese Weise nur langsam zerstört.

Haben zudem die von Null verschiedenen Einträge in der zu eliminierenden Zeile alle den Wert ± 1, so sind die notwendigen Spaltenoperationen besonders einfach. In diesem Fall ist es nicht erforderlich, echte Vielfache einzelner Spalten zu anderen Spalten zu addieren. Dadurch wächst die Größe der von Null verschiedenen Matrixeinträge nur sehr langsam an.

Insgesamt wird so ein Verfahren zur Verkleinerung der Relationenmatrix konstruiert, das die beiden folgenden - sich gegenseitig bedingenden - Eigenschaften hat:

- Die dünn besetzte Struktur der Matrix bleibt lange erhalten, und die Größe der von Null verschiedenen Einträge wächst nur sehr langsam.
- Es werden wesentlich weniger und wesentlich einfachere Spaltentransformationen benötigt als bei gewöhnlicher Hermite-Reduktion.

Mit diesen Überlegungen ergibt sich die im folgenden dargestellte erste Version des Rahmenalgorithmus ELIMINATION. Hier wird lediglich die Grundidee des Verfahrens in Form eines Algorithmus formuliert. Der Algorithmus wird in den anschließenden Abschnitten sukzessive konkretisiert und weiterentwickelt.

Algorithmus 9.1 (Rahmenalgorithmus ELIMINATION (1. Fassung))

Eingabe: das Relationensystem $\mathcal{R} = (F, A, n, m)$
Ausgabe: das Relationensystem $\mathcal{R}_1 = (F_1, B, n_1, m_1)$ mit $n_1 < n$, $F_1 \subset F$ und $\operatorname{Det} \Lambda(B) = \operatorname{Det} \Lambda(A)$

Setze $n_1 \leftarrow n$, $m_1 \leftarrow m$, $F_1 \leftarrow F$ und $B \leftarrow A$.		(1)
	Wähle einen Index $i \in [\![1, n_1]\!]$, so daß in der i-ten Zeile der Matrix B die Eliminationsbedingung erfüllt ist bzw. durch wenige einfache Spaltenoperationen erfüllt werden kann.	(2)
BREAKIF es existiert keine solche Zeile.		(3)
	Führe die notwendigen Spaltenoperationen auf der Matrix B durch. Es sei nun $j \in [\![1, m_1]\!]$ derjenige eindeutig bestimmte Index, für den $b_{ij} = \pm 1$ ist.	(4)
	Streiche Zeile i sowie Spalte j aus der Matrix B.	(5)
	Streiche das i-te Element aus der Formenbasis F_1.	(6)
	Vermindere n_1 und m_1 jeweils um 1.	(7)

9.2 Erweiterung der Aufgabenstellung

Bei der Konkretisierung des Algorithmus ELIMINATION ist ein wichtiger, bisher noch nicht erwähnter Zusammenhang zu den Schritten (7) und (12) des Algorithmus CLASSGROUP zu beachten. In diesen Schritten kann es, wie in Abschnitt 6.7 erläutert wurde, notwendig sein, zusätzliche Relationen über der verkleinerten Formenbasis F_1 zu generieren.

Nun ist es mit dem in Kapitel 8 beschriebenen Verfahren nicht möglich, Relationen über einer kleinen Formenbasis F_1 zu generieren, die eine beliebige Teilmenge von F ist. Der Grund hierfür ist die Tatsache, daß die Formenbasis F_1 keine vollständige Liste von Primformen (p, b_p) mit Primzahlen p unterhalb einer gewissen Schranke ist, sondern einige Primformen fehlen. Damit ist eine wichtige Voraussetzung für die Anwendung der Early-Abort-Strategie und der Large-Prime-Variante während der Generierung der Relationen nicht gegeben. Da zudem die Formenbasis F_1 sehr viel kleiner ist als die Formenbasis F, wäre die Erfolgswahrscheinlichkeit eines Verfahrens, das Relationen über F_1 generiert, in der Praxis viel zu gering.

Neue Relationen können also nur über der ursprünglichen Formenbasis F generiert werden. Sie müssen anschließend in Relationen über der Formenbasis F_1 umgerechnet werden. Wir erläutern im folgenden, wie diese Umrechnung erfolgt und welche Daten dazu benötigt werden. Diese Informationen müssen dann zusätzlich vom Algorithmus ELIMINATION berechnet und ausgegeben werden.

Im Verlauf des Algorithmus ELIMINATION wird iterativ eine Folge

$$A = A^{(0)}, A^{(1)}, \ldots, A^{(n-n_1)} = B$$

von Matrizen und eine Folge

$$F = F^{(0)} \supset F^{(1)} \supset \ldots \supset F^{(n-n_1)} = F_1$$

von Formenbasen berechnet, so daß die Spalten der Matrix $A^{(k)}$ Relationen über $F^{(k)}$ sind $(0 \leq k \leq n - n_1)$. Dabei ist

$$\begin{aligned} A^{(k)} &\in \mathbb{Z}^{(n-k)\times(m-k)} \qquad (0 \leq k \leq n - n_1), \\ \#F^{(k)} &= n - k \qquad (0 \leq k \leq n - n_1). \end{aligned}$$

Ferner existieren paarweise verschiedene Indizes $u_1, \ldots, u_{n-n_1} \in [\![1, n]\!]$, so daß

$$F^{(k)} = F^{(k-1)} \setminus \{f_{u_k}\} \qquad (1 \leq k \leq n - n_1).$$

Wir bezeichnen den Vektor $u = (u_1, \ldots, u_{n-n_1}) \in \mathbb{Z}^{n-n_1}$ als **Eliminationsvektor**. Er gibt an, in welcher Reihenfolge die einzelnen Spalten der Relationenmatrix gestrichen wurden.

Unterläßt man nun während der einzelnen Iterationsschritte im Algorithmus ELIMINATION die Streichung von Zeilen, so erhält man eine Folge

$$A = A^{(0)}, A^{(1)}, \ldots, A^{(n-n_1)}$$

von Matrizen mit

$$A^{(k)} \in \mathbb{Z}^{n\times(m-k)} \qquad (0 \leq k \leq n - n_1),$$

für die

$$a^{(k)}_{u_k,j} = 0 \qquad (1 \leq k \leq n - n_1,\ 1 \leq j \leq m - k)$$

ist. Alle Matrizen $A^{(k)}$ haben also n Zeilen, k dieser Zeilen enthalten jedoch nur Nullen. Die Spalten der Matrizen $A^{(k)}$ sind offensichtlich Relationen über der ursprünglichen Formenbasis F. Die Matrix B ergibt sich dann aus der Matrix $A^{(n-n_1)}$, indem man die Zeilen mit den Nummern $u_1, \ldots, u_{n-n_1}$ streicht.

Wir konstruieren nun eine Matrix $C \in \mathbb{Z}^{n\times(n-n_1)}$, indem wir für $1 \leq k \leq n - n_1$ die beim Übergang von $A^{(k-1)}$ nach $A^{(k)}$ gestrichene Spalte von $A^{(k-1)}$ als k-te Spalte von C übernehmen. Offensichtlich gilt

$$c_{u_k,k} = \pm 1 \qquad (1 \leq k \leq n - n_1), \tag{9.2}$$

$$c_{u_k,j} = 0 \qquad (1 \leq k < j \leq n - n_1), \tag{9.3}$$

und die Spalten der Matrix C sind Relationen über der Formenbasis F. Ferner multiplizieren wir einige der Spalten von C mit (-1), so daß über (9.2) hinaus sogar

$$c_{u_k,k} = 1 \qquad (1 \leq k \leq n - n_1) \tag{9.4}$$

gilt.

Unter Verwendung der Matrix C kann man nun mit Hilfe des im folgenden dargestellten Algorithmus eine beliebige Relation über der Formenbasis F in eine Relation über der Formenbasis F_1 umrechnen. Wir bezeichnen daher die Matrix C als **Umformungsmatrix** von F nach F_1.

Algorithmus 9.2 (Umrechnung von Relationen)

Eingabe:	die Parameter n und n_1, die Umformungsmatrix $C \in \mathbb{Z}^{n\times(n-n_1)}$ von F nach F_1, der zugehörige Eliminationsvektor $u \in \mathbb{Z}^{n-n_1}$ sowie eine Relation $w \in \mathbb{Z}^n$ über F
Ausgabe:	eine Relation $v \in \mathbb{Z}^{n_1}$ über F_1

FOR $k = 1$ TO $n - n_1$	(1)
Setze $w \leftarrow w - w_{u_k} \cdot C_i$.	(2)
Konstruiere durch Streichen der Komponenten mit den Nummern $u_1, \ldots, u_{n-n_1}$ aus dem Vektor w den Vektor $v \in \mathbb{Z}^{n_1}$.	(3)

Nach Beendigung der Schleife (1) - (2) gilt $w_{u_k} = 0$ für $1 \le k \le n - n_1$, und w ist eine Relation über F. Der Vektor v ist also eine Relation über F_1.

Diese Überlegung zeigt, daß der Algorithmus ELIMINATION neben der verkleinerten Relationenmatrix B und der verkleinerten Formenbasis F_1 auch die Umformungsmatrix C von F nach F_1 sowie den zugehörigen Eliminationsvektor u berechnen muß.

Wir führen ferner die Verkleinerung der Formenbasis nur implizit durch, indem wir einen Vektor $r \in \mathbb{Z}^{n_1}$ berechnen, der angibt, welche Elemente der ursprünglichen Formenbasis F in der neuen Formenbasis F_1 an welcher Position stehen. Ist also

$$r_i = j \qquad (1 \le i \le n_1)$$

mit $j \in [\![1, n]\!]$, so ist das i-te Elemente der neuen Formenbasis F_1 gleich dem j-ten Element der ursprünglichen Formenbasis F. Wir bezeichnen den Vektor r als **Positionenvektor**.

Eingabe für den Algorithmus ELIMINATION ist somit nur noch die Relationenmatrix $A \in \mathbb{Z}^{n\times m}$ mit $m > n$. Ausgegeben werden

- $n_1 \in [\![1, n]\!]$,
- die verkleinerte Relationenmatrix $B \in \mathbb{Z}^{n_1\times m_1}$, wobei $m_1 = m - (n - n_1)$ ist,
- der Positionenvektor $r \in \mathbb{Z}^{n_1}$, so daß die Spalten der Matrix B Relationen über der Formenbasis $F_1 = \{f_{r_1}, \ldots, f_{r_{n_1}}\}$ sind,
- die Umformungsmatrix $C \in \mathbb{Z}^{n\times(n-n_1)}$ von F nach F_1,
- der zugehörige Eliminationsvektor $u \in \mathbb{Z}^{n-n_1}$.

Algorithmus 9.3 (Rahmenalgorithmus ELIMINATION (2. Fassung))

Eingabe: die Relationenmatrix $A \in \mathbb{Z}^{n\times m}$
Ausgabe: n_1, B, r, C und u wie oben angegeben

Setze $m_1 \leftarrow m$, $k \leftarrow 0$ und $B \leftarrow A$.	(1)
Wähle eine Zeile i in B, in der die Eliminationsbedingung erfüllt ist bzw. durch wenige einfache Spaltenoperationen erfüllt werden kann.	(2)
BREAKIF es existiert keine solche Zeile.	(3)
Führe die notwendigen Spaltenoperationen auf der Matrix B durch. Es sei nun $j \in [\![1, m_1]\!]$ derjenige eindeutig bestimmte Index, für den $b_{ij} = \pm 1$ ist.	(4)
IF $b_{ij} = -1$ THEN setze (Spalte) $B_j \leftarrow (-1) \cdot B_j$.	(5)
Erhöhe k um 1 und setze $u_k \leftarrow i$.	(6)
Setze (Spalte) $C_k \leftarrow B_j$.	(7)
Streiche die Spalte B_j aus der Matrix B und vermindere m_1 um 1.	(8)
Streiche die Zeilen mit den Nummern $u_1, \ldots, u_k$ aus der Matrix B.	(9)
Konstruiere den Positionenvektor r. Setze dazu zunächst $r_i \leftarrow i$ für $1 \le i \le n$ und streiche dann die Komponenten mit den Nummern $u_1, \ldots, u_k$ aus r.	(10)
Setze $n_1 \leftarrow n - k$.	(11)

Die Übernahme von Spalten aus der Matrix B in die Matrix C sowie das Streichen von Spalten aus der Matrix B kann in der Praxis durch das Umsetzen von Zeigern realisiert werden. Das Kopieren der entsprechenden Matrixeinträge ist dazu nicht erforderlich.

9.3 Algorithmen zur Elimination

Im folgenden beschreiben wir die Auswahl der zu eliminierenden Zeilen und Spalten und die zur Herstellung der Eliminationsbedingung notwendigen Spaltenoperationen (Schritte (2) und (4) von Algorithmus 9.3). Wir formulieren dazu drei verschiedene Teilalgorithmen, die jeweils unterschiedlich aufwendige Spaltenoperationen zur Herstellung der Eliminationsbedingung zulassen. Die drei Teilalgorithmen werden später nacheinander ausgeführt, um eine möglichst weitgehende Verkleinerung der Relationenmatrix zu erzielen.

Aus technischen Gründen werden die Teilalgorithmen so formuliert, daß sie die gesamte Schleife (2) - (8) der bisherigen Version des Algorithmus ELIMINATION ersetzen. Aufgabe des Algorithmus ELIMINATION, dessen endgültige Version in

Abschnitt 9.5 angegeben wird, ist dann im wesentlichen nur noch die Steuerung dieser Teilalgorithmen.

Ein- und Ausgabedaten der Teilalgorithmen sind die Variablen B, C, u, n, m_1 und k aus dem Algorithmus ELIMINATION, die hier sukzessive verändert werden. Zu jedem Zeitpunkt gibt k die Anzahl der bereits eliminierten Spalten an, ferner ist $C \in \mathbb{Z}^{n \times k}$, $B \in \mathbb{Z}^{n \times (m_1)}$ und $m_1 = m - k$. Ferner ist zu jedem Zeitpunkt $u \in \mathbb{Z}^k$ der bei der Elimination der ersten k Spalten entstandene Eliminationsvektor.

Zur Formalisierung der in den einzelnen Teilalgorithmen für die Elimination zu erfüllenden Bedingungen setzen wir für eine Matrix $B \in \mathbb{Z}^{n \times m_1}$

$$N_B(i) = \begin{cases} 0 & \text{falls ein Index } j \in [\![1, m_1]\!] \\ & \text{existiert mit } |b_{ij}| > 1, \\ \sum\limits_{j=k+1}^{m} |b_{ij}| & \text{sonst.} \end{cases} \qquad (1 \le i \le n)$$

Der Wert $N_B(i)$ gibt also an, ob die i-te Zeile von B ausschließlich Einträge mit den Werten $0, +1, -1$ enthält, und, falls dies der Fall ist, wieviele Einträge in dieser Zeile den Wert $+1$ oder -1 haben. Man beachte, daß $N_B(i)$ nicht nur dann den Wert 0 hat, wenn die i-te Zeile von B einen Eintrag mit Betrag größer als 1 enthält, sondern auch dann, wenn alle Einträge der i-ten Zeile den Wert Null haben.

Im ersten Teilalgorithmus werden nun alle diejenigen Zeilen eliminiert, für die die Eliminationsbedingung (9.1) ohne vorherige Spaltenumformungen erfüllt ist. Offenbar liegt diese Situation für eine Zeile i genau dann vor, wenn $N_B(i) = 1$ ist.

Algorithmus 9.4 (Teilalgorithmus ELIM-1)

Eingabe: n, k, m_1, B, C, u wie oben angegeben
Ausgabe: geänderte Werte für k, m_1, B, C, u wie oben angegeben

WHILE es existiert ein Index $i \in [\![1, n]\!]$ mit $N_B(i) = 1$		(1)
	Es sei $j \in [\![1, m_1]\!]$ derjenige eindeutig bestimmte Index, für den $b_{ij} = \pm 1$ ist.	(2)
	IF $b_{ij} = -1$ THEN setze (Spalte) $B_j \leftarrow (-1) \cdot B_j$.	(3)
	Erhöhe k um 1 und setze $u_k \leftarrow i$.	(4)
	Setze (Spalte) $C_k \leftarrow B_j$.	(5)
	Streiche die Spalte B_j aus der Matrix B und vermindere m_1 um 1.	(6)

Man beachte, daß nach der Durchführung einiger Iterationen dieses Algorithmus die Bedingung $N_B(i) = 1$ für Indizes i erfüllt sein kann, für die sie zuvor nicht erfüllt war. Es reicht also nicht aus, die Matrix B einmal zeilenweise von oben nach unten daraufhin zu untersuchen, welche Zeilen und Spalten eliminiert werden können.

Die dünn besetzte Struktur der Relationenmatrix führt dazu, daß schon mit diesem einfachen Algorithmus eine große Anzahl von Zeilen und Spalten eliminiert werden kann. Genauere Daten sind der Tabelle am Ende dieses Kapitels zu entnehmen.

Nach der Durchführung von Algorithmus ELIM-1 enthält die Matrix B keine Zeile mehr, die die Eliminationsbedingung (9.1) erfüllt. Wir versuchen daher in den nächsten Schritten, mit elementaren Spaltenumformungen diese Situation herzustellen, ohne daß Anzahl und Größe der Matrixeinträge stark wachsen.

Dies ist leicht möglich, wenn in einer Zeile der Matrix genau zwei Einträge verschieden von Null sind und diese beiden Einträge den Wert ± 1 haben. Dann nämlich kann man durch eine einzige Addition bzw. Subtraktion zweier Spalten der Matrix die Eliminationsbedingung (9.1) erfüllen. Wir eliminieren also alle Zeilen i mit $N_B(i) = 2$.

Die Anzahl der von Null verschiedenen Einträge in der Matrix B vergrößert sich durch diese Eliminationen nicht. In einer der beiden Spalten kommen nämlich höchstens so viele von Null verschiedene Einträge hinzu wie die zweite Spalte enthält; diese wird aber anschließend aus der Matrix entfernt.

Auch die Größe der Matrixeinträge wächst nicht stark an, da nur Additionen bzw. Subtraktionen von Spalten durchgeführt werden, nicht jedoch Additionen bzw. Subtraktionen von echten Vielfachen von Spalten. Würde man zulassen, daß einer der beiden Einträge nicht den Betrag 1 hätte, so müßte man ein Vielfaches der anderen Spalte addieren bzw. subtrahieren, um diesen Eintrag zu eliminieren. Dabei würde die Größe der Matrixeinträge schneller anwachsen.

Auch hier gilt wie beim Algorithmus ELIM-1, daß durch die Elimination einzelner Zeilen und Spalten eventuell die Bedingung $N_B(i) = 2$ für andere Zeilen der Matrix hergestellt wird. Ferner ist es hier auch möglich, daß nach der Elimination einzelner Zeilen und Spalten andere Zeilen sogar die stärkere Bedingung $N_B(i) = 1$ erfüllen. Auch solche Zeilen sollen sofort eliminiert werden.

Algorithmus 9.5 (Teilalgorithmus ELIM-2)

Eingabe: n, k, m_1, B, C, u wie oben angegeben
Ausgabe: geänderte Werte für k, m_1, B, C, u wie oben angegeben

WHILE es existiert ein Index $i \in [\![1, n]\!]$ mit $N_B(i) \in \{0, 1\}$			(1)
	IF $N_B(i) = 1$		(2)
	THEN	Es sei $j \in [\![1, m_1]\!]$ derjenige eindeutig bestimmte Index, für den $b_{ij} = \pm 1$ ist.	(3)
	ELSE	Es seien $j, j' \in [\![1, n]\!]$ die beiden einzigen Indizes mit $b_{ij} = \pm 1$, $b_{ij'} = \pm 1$.	(4)
		Setze (Spalte) $B_{j'} \leftarrow B_{j'} - e \cdot B_j$, wobei $e = b_{ij} \cdot b_{ij'} = \pm 1$ gesetzt wird.	(5)
	IF $b_{ij} = -1$ THEN setze (Spalte) $B_j \leftarrow (-1) \cdot B_j$.		(6)
	Erhöhe k um 1 und setze $u_k \leftarrow i$.		(7)
	Setze (Spalte) $C_k \leftarrow B_j$.		(8)
	Streiche die Spalte B_j aus der Matrix B und vermindere m_1 um 1.		(9)

Algorithmus ELIM-2 ist eine Erweiterung des Algorithmus ELIM-1 und stimmt in wesentlichen Teilen mit diesem überein. Insofern stellt sich die Frage, warum man nicht gleich diesen Algorithmus verwendet, um simultan alle Zeilen mit einem oder zwei Einträgen aus der Matrix B zu eliminieren. Das zweistufige Vorgehen hat den Vorteil, daß die Matrix zunächst ohne Spaltenoperationen so weit wie möglich verkleinert wird, bevor zum ersten Mal Spaltenoperationen vorgenommen werden.

9.4 Strukturierte Elimination

Es liegt nun nahe, das prinzipielle Vorgehen von Algorithmus ELIM-2 für Zeilen mit mehr als zwei Einträgen weiterzuführen. Man würde dann sukzessive zunächst alle Zeilen mit drei Einträgen eliminieren, dann die mit vier Einträgen und so weiter. Die Elimination würde abgebrochen, wenn die zur Erfüllung der Eliminationsbedingung (9.1) notwendigen Spaltenoperationen zu aufwendig würden.

Diese Vorgehensweise hat sich aber in der Praxis als ungünstig erwiesen. Enthält nämlich eine Zeile mehr als zwei von Null verschiedene Einträge, so sind mindestens zwei Spaltenoperationen notwendig, um diese Zeile zu eliminieren. Die Anzahl der Einträge in der Matrix steigt daher an, und die dünn besetzte Struktur geht schnell verloren. Als Folge davon sind in den dann folgenden Spaltenoperationen fast alle Einträge der beteiligten Spalten besetzt, so daß auch die Größe der Einträge schnell anwächst.

Es ist aber dennoch möglich, die Verkleinerung der Relationenmatrix fortzusetzen, ohne daß die dünn besetzte Struktur zu schnell verloren geht. Dies geschieht mit Hilfe einer modifizierten Form der von Odlyzko und LaMacchia in [32] bzw. [27] vorgeschlagenen sogenannten **strukturierten Gauß-Elimination**.

Die Zeilen der Matrix B werden dazu in zwei Kategorien eingeteilt. "Schwer" sind solche Zeilen, die entweder viele von Null verschiedene Einträge enthalten oder in denen es Einträge mit Betrag größer als 1 gibt. "Leicht" sind dagegen solche Zeilen, die wenige von Null verschiedene Einträge enthalten, die zudem alle den Wert ± 1 haben. Schwere und leichte Zeilen können sich in der Matrix beliebig abwechseln.

Die Anzahl der in den leichten Zeilen maximal zugelassenen von Null verschiedenen Einträge wird als Parameter M in den Algorithmus eingegeben. Leicht sind also solche Zeilen i, für die $0 < N_B(i) \leq M$ ist.

Ferner definiert man für jede Spalte der Matrix B ein Gewicht w_j $(1 \leq j \leq m_1)$. Dieses Gewicht ist die Anzahl der Einträge in der entsprechenden Spalte, die in leichten Zeilen stehen.

Für die Elimination betrachten wir nun nur Spalten mit Gewicht 1. Solche Spalten enthalten in genau einer der leichten Zeilen einen Eintrag mit Wert ± 1, in allen anderen leichten Zeilen aber Nullen. In den schweren Zeilen solcher Spalten können beliebige Einträge vorliegen. Wir bezeichnen solche Spalten als **Reduktionsspalten**.

Es sei nun B_j eine solche Reduktionsspalte und i die Nummer der leichten Zeile, in der der oben erwähnte Eintrag steht. Mit Hilfe der Spalte B_j werden nun alle

Einträge in Zeile i zu Null gemacht. Dazu wird die Spalte B_j zu allen anderen Spalten, die in der Zeile i ebenfalls einen von Null verschiedenen Eintrag haben, addiert bzw. von diesen subtrahiert.

Man beachte, daß die hierbei veränderten Spalten nicht notwendigerweise selbst Reduktionsspalten sind. Man beachte ferner, daß durch die Spaltenoperationen die Gewichte der modifizierten Spalten abnehmen.

Schließlich enthält die Zeile i nur noch in der Reduktionsspalte einen von Null verschiedenen Eintrag, und die Eliminationsbedingung (9.1) ist erfüllt; Zeile i und Spalte j können also eliminiert werden.

Der entscheidende Vorteil dieser Vorgehensweise besteht darin, daß die Reduktionsspalte in keiner der leichten Zeilen außer einer einzigen einen von Null verschiedenen Eintrag hat. Bei den für die Elimination erforderlichen Spaltenoperationen entsteht daher in keiner der leichten Zeilen ein neuer von Null verschiedener Eintrag.

Algorithmus 9.6 (Teilalgorithmus ELIM-STRUKT)

Eingabe: n, k, m_1, B, C, u wie oben angegeben sowie $M \in [\![3, m_1]\!]$
Ausgabe: geänderte Werte für k, m_1, B, C, u wie oben angegeben

/∗ Einteilung in schwere und leichte Zeilen ∗/	
FOR $i = 1$ TO n	(1)
IF $0 < N(i,k) \le M$	(2)
THEN setze $s_i \leftarrow 1$.	(3)
ELSE setze $s_i \leftarrow 0$.	(4)
/∗ Initialisierung der Gewichte ∗/	
FOR $j = 1$ TO m_1	(5)
Setze $w_j \leftarrow \sum_{i=1}^{n} \lvert b_{ij}\rvert \cdot s_i$.	(6)
/∗ Elimination der Spalten mit Gewicht 1 ∗/	
WHILE es existiert ein Index $j \in [\![1, m_1]\!]$ mit $w_j = 1$	(7)
Es sei $i \in [\![1, n]\!]$ die Nummer der eindeutig bestimmten leichten Zeile, in der Spalte j einen Eintrag hat.	(8)
Es seien $j_1, \ldots, j_N$ $(N < M)$ die Nummern der übrigen Spalten, in denen die Einträge der Zeile i stehen.	(9)
FOR $\nu = 1$ TO N	(10)
Setze (Spalte) $B_{j_\nu} \leftarrow B_{j_\nu} - e \cdot B_j$, wobei $e = B_{ij} \cdot B_{i,j_\nu} = \pm 1$ gesetzt wird.	(11)
Vermindere w_{j_ν} um 1.	(12)
IF $b_{ij} = -1$ THEN setze (Spalte) $B_j \leftarrow (-1) \cdot B_j$.	(13)
Erhöhe k um 1 und setze $u_k \leftarrow i$.	(14)
Setze (Spalte) $C_k \leftarrow B_j$.	(15)
Streiche die Spalte B_j aus der Matrix B und vermindere m_1 um 1.	(16)
Streiche die Komponente mit der Nummer j aus dem Vektor w.	(17)

Der Algorithmus ELIM-STRUKT wird zunächst für $M = 20$ durchgeführt und dann sukzessive für abnehmendes M bis $M = 4$. Für jeden Wert von M wird dabei die Einteilung in leichte und schwere Zeilen neu vorgenommen.

Da M abnimmt, wird die Anzahl der leichten Zeilen in jedem Durchlauf des Algorithmus kleiner. Dadurch besteht die Möglichkeit, daß Spalten, die bei größerem M noch mehrere Einträge im Bereich der leichten Zeilen haben, später bei kleinerem M nur noch einen Eintrag im Bereich der leichten Zeilen haben und daher doch noch als Reduktionsspalten verwendet werden können.

Durch das hier beschriebene Vorgehen werden die Spaltenoperationen, die zur Herstellung der Eliminationsbedingung (9.1) führen, automatisch so gesteuert, daß Operationen mit geringer Auswirkung auf Anzahl und Größe der Matrixeinträge früher ausgeführt werden also solche, durch die die Anzahl oder die Größe der Matrixeinträge stärker ansteigt.

9.5 Die Endfassung des Algorithmus ELIMINATION

Zur Steuerung der in den letzten beiden Abschnitten formulierten Teilalgorithmen dient die im folgenden angegebene endgültige Version des Rahmenalgorithmus ELIMINATION:

Algorithmus 9.7 (Rahmenalgorithmus ELIMINATION (Endfassung))

Eingabe: die Relationenmatrix $A \in \mathbb{Z}^{n \times m}$
Ausgabe: n_1, B, r, C und u wie oben angegeben

Setze $m_1 \leftarrow m$, $k \leftarrow 0$ und $B \leftarrow A$.	(1)
/* Aufruf der Teilalgorithmen */	
Eliminiere aus B alle Zeilen i mit $N_B(i) = 1$. (Teilalgorithmus ELIM-1)	(2)
Eliminiere aus B alle Zeilen i mit $N_B(i) \in \{1, 2\}$. (Teilalgorithmus ELIM-2)	(3)
FOR $M = 20$ DOWNTO 4	(4)
Führe strukturierte Elimination mit dem Parameter M durch. (Teilalgorithmus ELIM-STRUKT)	(5)
/* Endbehandlung */	
Streiche die Zeilen mit den Nummern $u_1, \ldots, u_k$ aus der Matrix B.	(6)
Konstruiere den Positionenvektor r. Setze dazu zunächst $r_i \leftarrow i$ für $1 \leq i \leq n$ und streiche dann die Komponenten mit den Nummern $u_1, \ldots, u_k$ aus r.	(7)
Setze $n_1 \leftarrow n - k$.	(8)

In der Praxis liegt während des gesamten Vorgangs der Elimination die Relationenmatrix in einer speziellen Datenstruktur vor, auf deren Einzelheiten wir hier jedoch nicht eingehen. Diese Datenstruktur nutzt die dünne Besetzung der Matrix aus, indem nur die von Null verschiedenen Einträge und deren Positionen gespeichert werden.

Die Tabelle auf der folgenden Seite zeigt die Anzahl der mit den verschiedenen Teilalgorithmen eliminierten Zeilen und Spalten. Die Tabelle enthält die folgenden Daten:

- absolute Diskriminante $|\Delta|$,
- Anzahl n der Zeilen in der ursprünglichen Relationenmatrix A,
- Anzahl k_1 der mit dem Teilalgorithmus ELIM-1 eliminierten Zeilen,
- Anzahl k_2 der mit dem Teilalgorithmus ELIM-2 eliminierten Zeilen,
- Anzahl k_3 der in allen Durchläufen des Teilalgorithmus ELIM-STRUKT insgesamt eliminierten Zeilen,
- Anzahl n_1 der Zeilen der nach Ende der Elimination vorliegenden Relationenmatrix B.

Aus den angegebenen Daten wird deutlich, daß bereits eine sehr große Anzahl von Zeilen mit dem sehr einfachen Algorithmus ELIM-1 entfernt werden kann. Auch die Algorithmen ELIM-2 und ELIM-STRUKT haben noch eine beachtliche Verkleinerung zur Folge. Die Anzahl n_1 der Zeilen in der verkleinerten Relationenmatrix B liegt immer unterhalb von einem Zehntel der ursprünglichen Zeilenanzahl n.

Tabelle 9.8 (Verkleinerung der Relationenmatrix)

$\|\Delta\|$	n	k_1	k_2	k_3	n_1
$4 \cdot 10^{10} + 4$	249	210	20	10	9
$4 \cdot 10^{11} + 4$	300	247	30	13	10
$4 \cdot 10^{12} + 4$	339	258	49	25	7
$4 \cdot 10^{13} + 4$	385	275	64	35	11
$4 \cdot 10^{14} + 4$	459	362	56	31	10
$4 \cdot 10^{15} + 4$	483	319	95	56	13
$4 \cdot 10^{16} + 4$	556	424	75	40	17
$4 \cdot 10^{17} + 4$	613	402	119	72	20
$4 \cdot 10^{18} + 4$	649	377	156	94	22
$4 \cdot 10^{19} + 4$	708	481	115	88	24
$4 \cdot 10^{20} + 4$	796	517	160	92	27
$4 \cdot 10^{21} + 4$	896	522	199	142	33
$4 \cdot 10^{22} + 4$	953	566	209	141	37
$4 \cdot 10^{23} + 4$	1001	577	206	178	40
$4 \cdot 10^{24} + 4$	1131	595	248	237	51
$4 \cdot 10^{25} + 4$	1181	597	267	247	70
$4 \cdot 10^{26} + 4$	1330	688	300	282	60
$4 \cdot 10^{27} + 4$	1304	640	286	301	77
$4 \cdot 10^{28} + 4$	1441	692	351	325	73
$4 \cdot 10^{29} + 4$	1511	712	341	364	94
$4 \cdot 10^{30} + 4$	1604	765	349	400	90
$4 \cdot 10^{31} + 4$	1754	866	374	426	88
$4 \cdot 10^{32} + 4$	1860	870	400	463	127
$4 \cdot 10^{33} + 4$	1933	770	436	553	174
$4 \cdot 10^{34} + 4$	2015	884	485	508	133
$4 \cdot 10^{35} + 4$	2090	938	477	538	137
$4 \cdot 10^{36} + 4$	2238	933	493	631	181
$4 \cdot 10^{37} + 4$	2339	908	523	694	214
$4 \cdot 10^{38} + 4$	2439	995	549	687	208
$4 \cdot 10^{39} + 4$	2483	993	542	721	227
$4 \cdot 10^{40} + 4$	2695	1142	587	754	212
$4 \cdot 10^{41} + 4$	2761	1059	627	815	260
$4 \cdot 10^{42} + 4$	2939	1093	674	887	285
$4 \cdot 10^{43} + 4$	3112	1199	700	910	303
$4 \cdot 10^{44} + 4$	3216	1224	725	981	286
$4 \cdot 10^{45} + 4$	3314	1259	708	1024	323

Kapitel 10

Die Berechnung des Moduls

In diesem Kapitel beschreiben wir die Schritte (5) bis (7) des Algorithmus CLASS-GROUP. Eingabe für diesen Teil des Algorithmus ist die zuvor in Schritt (4) konstruierte Matrix $B \in \mathbb{Z}^{n_1 \times m_1}$ mit $m_1 > n_1$, deren Spalten Relationen über der verkleinerten Formenbasis F_1 sind.

Das Ziel der Schritte (5) bis (7) besteht darin, ein möglichst kleines Vielfaches d der Determinante D des Gitters $\Lambda(B)$ zu bestimmen. Dieses Vielfache d von D wird dann im nachfolgenden Schritt (8) als Modul für die Hermite-Reduktion der Matrix B verwendet. Die grundsätzliche Idee zur Berechnung von d wurde bereits in Abschnitt 6.6 erläutert. Sie besteht darin, zwei verschiedene reguläre Teilmatrizen $B', B'' \in \mathbb{Z}^{n_1 \times n_1}$ von B auszuwählen, die Determinanten d' und d'' dieser Teilmatrizen zu berechnen und dann $d = \gcd(d', d'')$ zu setzen.

Ist die Matrix B singulär, so muß sie vor der Berechnung von d zunächst durch zusätzliche Relationen über F_1 so erweitert werden, daß sie regulär ist. Da eine eventuelle Singularität von B nicht in allen Fällen mit einfachen Mitteln zu erkennen ist, muß man in Kauf nehmen, daß sich die Notwendigkeit der Generierung zusätzlicher Relationen in einigen Fällen erst im Laufe des Versuchs, die regulären Teilmatrizen B' und B'' auszuwählen, herausstellt.

10.1 Vorüberlegungen

In Kapitel 6 wurde bereits erwähnt, daß die Matrix B nicht nur regulär sein, sondern auch mindestens $(n_1 + 1)$ von Null verschiedene Spalten enthalten muß, damit die Auswahl zweier verschiedener regulärer n_1-reihiger Teilmatrizen B' und B'' möglich ist.

Diese Bedingung ist offensichtlich notwendig. Falls nämlich B nur n_1 von Null verschiedene Spalten enthält, so kann man zwar eine reguläre Teilmatrix $B' \in \mathbb{Z}^{n_1 \times n_1}$ von B auswählen, alle nicht zu B' gehörenden Spalten von B sind aber Null, so daß man keine von B' verschiedene Teilmatrix auswählen kann. Wir zeigen nun, daß die oben genannte Bedingung tatsächlich auch hinreichend ist.

Satz 10.1 *Es sei $M \in \mathbb{Z}^{\nu \times \mu}$ mit $\mu > \nu$ regulär. Dann gilt: Falls M mindestens $(\nu+1)$ von Null verschiedene Spalten enthält, so existieren zwei verschiedene ν-Tupel $(i_1, \ldots, i_\nu)$ und $(j_1, \ldots, j_\nu)$ mit $1 \leq i_1 < \ldots < i_\nu \leq \mu$ und $1 \leq j_1 < \ldots < j_\nu \leq \mu$, so daß die beiden Teilmatrizen M' und M'' von M, die aus den Spalten $M_{i_1}, \ldots, M_{i_\nu}$ bzw. $M_{j_1}, \ldots, M_{j_\nu}$ bestehen, regulär sind.*

Beweis: Da M regulär ist, existiert sicher eine reguläre Teilmatrix $M' \in \mathbb{Z}^{\nu \times \nu}$. O.B.d.A. bestehe M' aus den ersten ν Spalten von M. Dann sind die Vektoren der Menge $V = \{M_1, \ldots, M_\nu\}$ offenbar linear unabhängig. Da nun M mindestens $(\nu + 1)$ von Null verschiedene Spalten enthält, existiert ein Index $\kappa \in [\![\nu + 1, \mu]\!]$, so daß M_j nicht der Nullvektor ist. Nach dem Basisergänzungssatz läßt sich dann die Menge $W = \{M_\kappa\}$ mit geeigneten Vektoren aus V zu einer ν-elementigen Menge W' linear unabhängiger Vektoren ergänzen. Die Teilmatrix M'' von M, deren Spalten die Vektoren aus W' sind, ist dann offensichtlich regulär. ■

In der Praxis zeigt sich, daß die Matrix B, falls sie regulär ist, in der Regel auch mehr als n_1 von Null verschiedene Spalten enthält, also die Voraussetzungen des Satzes 10.1 erfüllt. Ist dies ausnahmsweise einmal nicht der Fall, so muß die Matrix B, obwohl sie regulär ist, vor der Auswahl der Teilmatrizen durch zusätzliche Relationen erweitert werden.

Mit der Auswahl der beiden Teilmatrizen B' und B'' von B ist das Ziel verbunden, ein möglichst kleines Vielfaches d von $D = \text{Det } \Lambda(B)$ zu ermitteln. Der größte gemeinsame Teiler d der Determinanten d' und d'' von B' und B'' soll also möglichst klein sein. Nun kann man zwar vor der Berechnung von d' und d'' keine Aussagen über die Größe von $d = \gcd(d', d'')$ machen, man kann jedoch die Auswahl der beiden Teilmatrizen so steuern, daß d mit möglichst hoher Wahrscheinlichkeit klein ist. Dazu versucht man zu erreichen, daß B'' möglichst viele der nicht zu B' gehörenden Spalten von B enthält. Dann unterscheiden sich mit hoher Wahrscheinlichkeit die von den Spalten der beiden Teilmatrizen erzeugten Teilgitter von $\Lambda(B)$ sehr stark, und d' und d'' enthalten - mit Ausnahme der in D enthaltenen Primfaktoren - nicht viele gemeinsame Primteiler. Man beachte in diesem Zusammenhang, daß es nicht in jedem Fall möglich ist, alle nicht zu B' gehörenden und von Null verschiedenen Spalten von B in die Matrix B'' zu integrieren, da unter diesen Spalten lineare Abhängigkeiten auftreten können.

Man beachte ferner, daß die beschriebene Vorgehensweise keine Garantie dafür bietet, daß d sehr viel kleiner ist als d' und d''. Es kann sogar nicht einmal garantiert werden, daß $|d'|$ und $|d''|$ tatsächlich verschieden sind. So ist es z.B. möglich, daß zwei Teilmatrizen B' und B'', die sich in einigen Spalten unterscheiden, trotzdem dasselbe Teilgitter von $\Lambda(B)$ erzeugen. Ferner können auch zwei unterschiedliche Teilgitter des Gitters $\Lambda(B)$ durchaus dieselbe Determinante haben. Man betrachte dazu als Beispiel die Matrix

$$B = \begin{pmatrix} 3 & 2 & 0 \\ 0 & 2 & 3 \end{pmatrix},$$

aus der die beiden verschiedenen Teilmatrizen

$$B' = \begin{pmatrix} 3 & 2 \\ 0 & 2 \end{pmatrix}, \quad B'' = \begin{pmatrix} 2 & 0 \\ 2 & 3 \end{pmatrix}$$

ausgewählt werden können. Offensichtlich ist dann $d = d' = d'' = 6$, eine Basis von $\Lambda(B)$ ist aber

$$\overline{B} = \begin{pmatrix} 1 & 0 \\ -2 & 1 \end{pmatrix},$$

d.h. es ist Det $\Lambda(B) = 1$. Bei anderer Auswahl der Teilmatrizen würde sich ein kleinerer Wert für d ergeben, dies ist aber zum Zeitpunkt der Konstruktion der Teilmatrizen nicht absehbar.

In der Praxis sind Probleme dieser Art nie aufgetreten. Vielmehr hat sich das Verfahren in allen Fällen als äußerst wirkungsvoll erwiesen. Obwohl in unserer Anwendung die Matrix B in der Regel nur etwa fünf Spalten mehr als Zeilen beinhaltet, die beiden Teilmatrizen B' und B'' sich also nicht um sehr viele Spalten unterscheiden können, ist in den meisten Fällen $d < 10 \cdot D$. Die Werte von d' und d'' liegen dabei für kleine Diskriminanten etwa bei D^2, für große Diskriminanten übersteigen sie D^{10}. Genauere Angaben hierzu können der Tabelle in Abschnitt 10.2 entnommen werden.

Wir untersuchen nun noch einen Spezialfall, der sich ergeben kann, wenn zwei Spalten der Matrix B übereinstimmen. In diesem Fall lassen sich zwei reguläre n_1-reihige Teilmatrizen B' und B'' von B konstruieren, die entweder gleich sind oder durch eine Permutation ihrer Spalten ineinander übergehen. B' und B'' erzeugen dann mit Sicherheit dasselbe Gitter, d.h. es ist $|d'| = |d''|$. Diesen Fall könnte man ausschließen, wenn man zusätzlich fordern würde, daß B mindestens $(n_1 + 1)$ paarweise verschiedene Spalten enthält. Die Überprüfung einer solchen Bedingung wäre jedoch sehr aufwendig, da man die Spalten von B paarweise miteinander vergleichen müßte. Außerdem kann man, wie sich aus den vorhergehenden Überlegungen ergibt, selbst dann, wenn die Spalten von B paarweise verschieden sind, nicht garantieren, daß $|d'| \neq |d''|$ ist. In der Praxis ist zudem, wie bereits erwähnt wurde, der Fall $|d'| = |d''|$ nie aufgetreten. Wir können daher auf die Einführung dieser Bedingung verzichten.

10.2 Der Gesamtablauf

Zur Realisierung der Schritte (5) bis (7) des Algorithmus CLASSGROUP dient der Rahmenalgorithmus MODUL, den wir in diesem Abschnitt formulieren. Eingabe für den Algorithmus MODUL ist, wie zu Beginn dieses Kapitels bereits erwähnt wurde, die zuvor in Schritt (4) des Algorithmus CLASSGROUP konstruierte Matrix $B \in \mathbb{Z}^{n_1 \times m_1}$ mit $m_1 > n_1$, deren Spalten Relationen über der Formenbasis F_1 sind.

Die Aufgabe des Algorithmus MODUL wurde ebenfalls bereits umrissen. Es sollen zwei verschiedene reguläre n_1-reihige Teilmatrizen B' und B'' von B ausgewählt, deren Determinanten d' und d'' berechnet und $d = \gcd(d', d'')$ ermittelt werden. Falls die Matrix B singulär ist oder nicht mindestens $(n_1 + 1)$ von Null verschiedene

Spalten enthält, muß sie vor der Berechnung von d durch zusätzliche Relationen über F_1 so erweitert werden, daß sie regulär ist und genügend viele nicht triviale Spalten enthält.

Bei der Formulierung des Algorithmus ist zu beachten, daß eine eventuelle Singularität von B mit möglichst einfachen Mitteln erkannt werden soll. Der Beweis der Regularität ist jedoch, wie bereits erwähnt wurde, nicht ohne großen Aufwand zu realisieren. Es ist also die Möglichkeit zu berücksichtigen, daß sich die Singularität von B eventuell erst im Laufe des Versuchs der Konstruktion der regulären Teilmatrizen herausstellt.

Wir geben nun zunächst den Algorithmus MODUL an und erläutern anschließend die einzelnen Schritte.

Algorithmus 10.2 (Rahmenalgorithmus MODUL)

Eingabe: die Relationenmatrix $B \in \mathbb{Z}^{n_1 \times m_1}$ (über F_1) mit $m_1 > n_1$

Ausgabe: die - eventuell veränderte - reguläre Relationenmatrix $B \in \mathbb{Z}^{n_1 \times m_1}$ (über F_1) mit $m_1 > n_1$ sowie ein möglichst kleines Vielfaches d von $D = \text{Det}\ \Lambda(B)$

Entferne alle Spalten aus B, die nur Nullen enthalten, und vermindere m_1 entsprechend.		(1)
IF $m_1 \leq n_1$		(2)
THEN	Generiere $a = n_1 - m_1 + 5$ neue Relationen $v_1, \ldots, v_a$ über F_1.	(3)
	Füge die Vektoren $v_1, \ldots, v_a$ als zusätzliche Spalten an die Matrix B an und erhöhe m_1 um a.	(4)
WHILE B enthält eine Zeile, die nur aus Nullen besteht		(5)
	Es sei i die Nummer dieser Zeile.	(6)
	Generiere eine neue Relation v über F_1. Versuche dabei zu erreichen, daß der i-te Eintrag von v nicht Null ist.	(7)
	Füge den Vektor v als zusätzliche Spalte an die Matrix B an und erhöhe m_1 um 1.	(8)
/* Nun ist nicht mehr auf einfache Weise feststellbar, ob B singulär ist. */		
	Versuche, zwei verschiedene reguläre Teilmatrizen $B', B'' \in \mathbb{Z}^{n_1 \times n_1}$ von B auszuwählen. Falls dies gelingt, berechne mit Hilfe einer modifizierten Form der modularen Gauß-Elimination simultan die Determinanten d' und d'' von B' und B'' und setze $d \leftarrow \gcd(d', d'')$. Andernfalls setze $d \leftarrow 0$. (Teilalgorithmus DETERMINANTE)	(9)
BREAKIF $d \neq 0$.		(10)
	Generiere fünf neue Relationen $v_1, \ldots, v_5$ über F_1.	(11)
	Füge die Vektoren $v_1, \ldots, v_5$ als zusätzliche Spalten an die Matrix B an und erhöhe m_1 um 5.	(12)

In Schritt (1) werden alle Spalten aus B entfernt, die nur aus Nullen bestehen. Enthält die Matrix B anschließend weniger als n_1 Spalten, so ist sie mit Sicherheit singulär. Enthält sie genau n_1 Spalten, so kann zwar keine Aussage darüber gemacht werden, ob sie regulär oder singulär ist; die im vorigen Abschnitt erläuterte Zusatzbedingung zur Auswahl der Teilmatrizen ist jedoch mit Sicherheit verletzt. In beiden Fällen muß also die Matrix B durch zusätzliche Relationen über der Formenbasis F_1 erweitert werden. Die Anzahl der neu zu generierenden Relationen wird dabei so gewählt, daß die Matrix B anschließend $(n_1 + 5)$ Spalten enthält.

Nun wird untersucht, ob B Zeilen enthält, die nur aus Nullen bestehen. Auch in diesem Fall ist B mit Sicherheit singulär, es werden also ebenfalls neue Relationen generiert. Dieser Vorgang wird so lange wiederholt, bis jede Zeile der Matrix B mindestens einen von Null verschiedenen Eintrag enthält.

Nach Beendigung der Schritte (1) bis (8) enthält die Matrix B mindestens $(n_1 + 1)$ Spalten, und keine der Zeilen oder Spalten von B besteht nur aus Nullen. Nun kann nicht mehr mit so einfachen Mitteln wie in den Schritten (2) oder (5) festgestellt werden, ob B regulär oder singulär ist. Der Beweis der Singularität bzw. Regularität kann unter Umständen ebenso aufwendig sein wie die Berechnung der Determinante einer n_1-reihigen Teilmatrix von B.

Daher wird in Schritt (9) der Teilalgorithmus DETERMINANTE durchlaufen, in dem zunächst versucht wird, die beiden Teilmatrizen B' und B'' von B zu konstruieren. Ist die Matrix B regulär, so gelingt die Konstruktion von B' und B'', da bereits sichergestellt ist, daß B mehr als n_1 von Null verschiedene Spalten enthält. In diesem Fall wird im weiteren Verlauf des Teilalgorithmus DETERMINANTE die Zahl $d = \gcd(\det B', \det B'')$ berechnet. Ist die Matrix B jedoch singulär, so schlägt bereits die Konstruktion der ersten Teilmatrix fehl. In diesem Fall gibt der Teilalgorithmus DETERMINANTE den Wert $d = 0$; die Matrix B muß dann erneut durch zusätzliche Relationen erweitert werden. Dies geschieht in den Schritten (11) und (12). Um anschließend zu entscheiden, ob genügend neue Relationen generiert wurden, muß Schritt (9) mit der erweiterten Matrix B erneut durchlaufen werden.

Die Schleife (9) - (12) muß eventuell mehrfach durchlaufen werden, bevor in Schritt (9) die Auswahl von B' und B'' möglich ist. Da Schritt (9) relativ aufwendig ist, soll die Anzahl der Durchläufe möglichst gering gehalten werden. Daher generieren wir in Schritt (11) nicht nur jeweils eine, sondern fünf neue Relationen, bevor wir Schritt (9) erneut durchführen.

Die Generierung zusätzlicher Relationen über der Formenbasis F_1 in den Schritten (3), (7) und (11) des Algorithmus MODUL wird mit dem Teilalgorithmus REL-ZUSATZ durchgeführt, den wir in Abschnitt 10.5 angeben.

Die folgende Tabelle zeigt, für welche der 36 Diskriminanten, die in den vorherigen Kapiteln als Beispiele verwendet wurden, im Verlauf des Algorithmus MODUL die Generierung zusätzlicher Relationen notwendig war. Dabei gibt a_1 die Anzahl der in Schritt (3) und a_2 die Anzahl der in Schritt (7) generierten Relationen an. Man beachte, daß für keine der 36 sonst betrachteten Diskriminanten die Schritte (11) und (12) durchlaufen wurden.

Tabelle 10.3 (Zusätzliche Relationen im Algorithmus MODUL)

$\|\Delta\|$	a_1	a_2
$4 \cdot 10^{30} + 4$	1	–
$4 \cdot 10^{33} + 4$	1	23

$\|\Delta\|$	a_1	a_2
$4 \cdot 10^{37} + 4$	–	5
$4 \cdot 10^{39} + 4$	–	13

$\|\Delta\|$	a_1	a_2
$4 \cdot 10^{41} + 4$	–	6
$4 \cdot 10^{42} + 4$	–	16

Die Generierung neuer Relationen während der Matrixreduktion ist also nur für sechs der insgesamt 36 sonst betrachteten Diskriminanten erforderlich. In allen anderen Fällen ist die Matrix B bereits bei Eingabe in den Algorithmus MODUL regulär und läßt die Auswahl regulärer Teilmatrizen zu. Hier wird deutlich, daß es mit der in Schritt (3) des Algorithmus CLASSGROUP verwendeten Strategie tatsächlich in fast allen Fällen gelingt, eine reguläre Relationenmatrix zu konstruieren.

Die nächste Tabelle enthält einige Ergebnisse, die mit dem Algorithmus MODUL erzielt wurden. Diese Ergebnisse zeigen den Erfolg unserer Vorgehensweise zur Verkleinerung des Vielfachen d von $D = \text{Det}\,\Lambda(B)$. In der zweiten Spalte der Tabelle sind die Größenordnungen $\overline{d'}$ und $\overline{d''}$ für die Determinanten d' und d'' der beiden Teilmatrizen angegeben. Die dritte Spalte enthält entsprechend die Größenordnung $\overline{d}$ des größten gemeinsamen Teilers d von d' und d''. Die angegebenen Daten zeigen, daß sich der Modul d in der Praxis tatsächlich sehr stark verkleinern läßt. Die vierte Spalte gibt den Faktor $d^* = d/D$ an, um den sich das Vielfache d noch von der tatsächlichen Determinante D des Gitters $\Lambda(B)$ unterscheidet. Hier wird deutlich, daß in den meisten Fällen d bereits mit D übereinstimmt oder sich von D nur noch höchstens um den Faktor 10 unterscheidet.

Tabelle 10.4 (Verkleinerung des Moduls d)

$\|\Delta\|$	$\overline{d'}, \overline{d''}$	$\overline{d}$	d^*
$4 \cdot 10^{10} + 4$	$10^{11}, 10^{13}$	10^{6}	2
$4 \cdot 10^{11} + 4$	$10^{10}, 10^{10}$	10^{7}	4
$4 \cdot 10^{12} + 4$	$10^{10}, 10^{11}$	10^{7}	6
$4 \cdot 10^{13} + 4$	$10^{14}, 10^{14}$	10^{7}	1
$4 \cdot 10^{14} + 4$	$10^{15}, 10^{16}$	10^{8}	2
$4 \cdot 10^{15} + 4$	$10^{16}, 10^{16}$	10^{10}	48
$4 \cdot 10^{16} + 4$	$10^{22}, 10^{23}$	10^{9}	1
$4 \cdot 10^{17} + 4$	$10^{20}, 10^{21}$	10^{10}	4
$4 \cdot 10^{18} + 4$	$10^{23}, 10^{23}$	10^{12}	42
$4 \cdot 10^{19} + 4$	$10^{24}, 10^{25}$	10^{11}	29
$4 \cdot 10^{20} + 4$	$10^{31}, 10^{32}$	10^{11}	1
$4 \cdot 10^{21} + 4$	$10^{36}, 10^{37}$	10^{11}	1
$4 \cdot 10^{22} + 4$	$10^{39}, 10^{41}$	10^{12}	1
$4 \cdot 10^{23} + 4$	$10^{41}, 10^{42}$	10^{12}	2
$4 \cdot 10^{24} + 4$	$10^{51}, 10^{51}$	10^{13}	2
$4 \cdot 10^{25} + 4$	$10^{63}, 10^{64}$	10^{14}	17
$4 \cdot 10^{26} + 4$	$10^{63}, 10^{66}$	10^{14}	1
$4 \cdot 10^{27} + 4$	$10^{70}, 10^{72}$	10^{15}	2

$\|\Delta\|$	$\overline{d'}, \overline{d''}$	$\overline{d}$	d^*
$4 \cdot 10^{28} + 4$	$10^{77}, 10^{78}$	10^{15}	1
$4 \cdot 10^{29} + 4$	$10^{89}, 10^{90}$	10^{15}	1
$4 \cdot 10^{30} + 4$	$10^{92}, 10^{93}$	10^{16}	6
$4 \cdot 10^{31} + 4$	$10^{99}, 10^{100}$	10^{16}	1
$4 \cdot 10^{32} + 4$	$10^{110}, 10^{111}$	10^{17}	2
$4 \cdot 10^{33} + 4$	$10^{131}, 10^{132}$	10^{18}	5
$4 \cdot 10^{34} + 4$	$10^{132}, 10^{133}$	10^{19}	8
$4 \cdot 10^{35} + 4$	$10^{135}, 10^{135}$	10^{19}	2
$4 \cdot 10^{36} + 4$	$10^{159}, 10^{159}$	10^{19}	1
$4 \cdot 10^{37} + 4$	$10^{180}, 10^{180}$	10^{20}	2
$4 \cdot 10^{38} + 4$	$10^{179}, 10^{179}$	10^{20}	1
$4 \cdot 10^{39} + 4$	$10^{191}, 10^{191}$	10^{20}	1
$4 \cdot 10^{40} + 4$	$10^{203}, 10^{204}$	10^{22}	4
$4 \cdot 10^{41} + 4$	$10^{217}, 10^{217}$	10^{21}	2
$4 \cdot 10^{42} + 4$	$10^{233}, 10^{233}$	10^{22}	4
$4 \cdot 10^{43} + 4$	$10^{251}, 10^{252}$	10^{22}	1
$4 \cdot 10^{44} + 4$	$10^{259}, 10^{260}$	10^{23}	2
$4 \cdot 10^{45} + 4$	$10^{276}, 10^{279}$	10^{24}	12

Wir geben nun die erste Fassung des Teilalgorithmus DETERMINANTE an, mit dem Schritt (9) des Algorithmus MODUL realisiert wird. Diese Fassung ergibt sich unmittelbar aus den Erläuterungen, die im Anschluß an die Darstellung des Algorithmus MODUL gegeben wurden.

Algorithmus 10.5 (Teilalgorithmus DETERMINANTE (1. Fassung))

Eingabe: eine Matrix $B \in \mathbb{Z}^{n_1 \times m_1}$ mit $m_1 > n_1$, bei der keine Spalte nur aus Nullen besteht

Ausgabe: falls B regulär ist, ein Vielfaches d von $\mathrm{Det}\,\Lambda(B)$, sonst $d = 0$

Versuche, eine reguläre Teilmatrix $B' \in \mathbb{Z}^{n_1 \times n_1}$ von B auszuwählen.	(1)
IF dies ist nicht möglich THEN setze $d \leftarrow 0$ und EXIT.	(2)
Wähle eine zweite reguläre Teilmatrix $B'' \in \mathbb{Z}^{n_1 \times n_1}$ von B aus, die von B' verschieden ist.	(3)
Berechne mit Hilfe einer modifizierten Form der modularen Gauß-Elimination simultan die Determinanten d' und d'' von B' und B''.	(4)
Setze $d \leftarrow \gcd(d', d'')$.	(5)

In den folgenden beiden Abschnitten werden die Schritte (1), (3) und (4) des Teilalgorithmus DETERMINANTE ausführlich beschrieben. Dabei wird der Algorithmus sukzessive weiterentwickelt und präzisiert.

10.3 Simultane Gauß-Elimination

In diesem Abschnitt erläutern wir die Berechnung der beiden Determinanten d' und d'' in Schritt (4) von Algorithmus 10.5. Wir setzen also voraus, daß die Auswahl der beiden Teilmatrizen B' und B'' bereits erfolgt ist.

Die Berechnung von d' und d'' soll nun mit der Technik der modularen Gauß-Elimination erfolgen. Zusätzlich soll die Tatsache ausgenutzt werden, daß in unserer Anwendung m_1 nicht sehr viel größer ist als n_1, sich die beiden Matrizen B' und B'' also nur in sehr wenigen Spalten unterscheiden. Dazu modifizieren wir den üblichen Algorithmus zur Gauß-Elimination (Algorithmus 3.31) und bestimmen für jede Primzahl p, für die die Gauß-Elimination durchgeführt werden muß, simultan $d'_p = \det B' \bmod p$ und $d''_p = \det B'' \bmod p$. Wir bezeichnen diesen Vorgang als **simultane Gauß-Elimination**.

Bei der Formulierung des Algorithmus zur simultanen Gauß-Elimination nehmen wir an, daß die Spalten der Matrix B so angeordnet sind, daß

- die Matrix B' aus den Spalten $B_1, \ldots, B_{n_1}$ besteht,
- die Matrix B'' aus den Spalten $B_1, \ldots, B_{n_1-k}, B_{n_1+1}, \ldots, B_{n_1+k}$ besteht.

Dabei ist $k \in [\![1, m_1 - n_1]\!]$ ein Parameter, der bei der Auswahl der zweiten Teilmatrix in Schritt (3) von Algorithmus 10.5 festgelegt wird. Man beachte, daß - wie bereits erwähnt - nicht in allen Fällen $k = m_1 - n_1$ gesetzt werden kann, da unter den Spalten $B_{n_1+1}, \ldots, B_{m_1}$ lineare Abhängigkeiten vorhanden sein können. Genauere Erläuterungen zur Festlegung von k werden im nächsten Abschnitt im Zusammenhang mit der Beschreibung des Algorithmus zur Auswahl der zweiten Teilmatrix gegeben.

Algorithmus 10.6 (Teilalgorithmus GAUSS-SIM)

Eingabe: eine reguläre Matrix $B \in \mathbb{Z}^{n_1 \times m_1}$ mit $m_1 > n_1$, bei der keine Spalte nur aus Nullen besteht, ein Parameter k, der, wie oben angegeben, die Teilmatrizen B' und B'' definiert, sowie eine Primzahl p

Ausgabe: $d'_p = \det B' \bmod p$ und $d''_p = \det B'' \bmod p$

Setze $d_p \leftarrow 1$.				(1)
Setze $c_{ij} \leftarrow b_{ij} \bmod p$ für $1 \leq i \leq n_1,\ \ 1 \leq j \leq n_1 + k$ und konstruiere so eine Matrix $C \in (\mathbb{Z}/p\mathbb{Z})^{n_1 \times (n_1+k)}$.				(2)
FOR $\ i = 1$ TO $n_1 - k$				(3)
	IF $\ c_{ii} = 0$			(4)
	THEN	Vertausche eine der Zeilen $i+1, \ldots, n_1$ von C mit der Zeile i, so daß dann $c_{ii} \neq 0$ ist, und setze $d_p \leftarrow p - d_p$.		(5)
		IF keine solche Zeile existiert THEN setze $d_p \leftarrow 0$ und EXIT.		(6)
	Berechne mit dem Euklidischen Algorithmus $x = (c_{ii})^{-1} \bmod p$.			(7)
	FOR $\ j = i + 1$ TO $n_1 + k$			(8)
		Setze (Spalte) $C_j \leftarrow C_j - (x \cdot c_{ij}) \cdot C_i$.		(9)
Setze $d_p \leftarrow d_p \cdot \prod_{i=1}^{n_1-k} c_{ii} \bmod p$.				(10)
Berechne mit üblicher Gauß-Elimination die Determinante d'_p (mod p) der Matrix $R' \in (\mathbb{Z}/p\mathbb{Z})^{k \times k}$, die entsteht, wenn man aus C die Spalten $1, \ldots, n_1 - k$ und $n_1+1, \ldots, n_1+k$ sowie die Zeilen $1, \ldots, n_1 - k$ streicht.				(11)
Berechne mit üblicher Gauß-Elimination die Determinante d''_p (mod p) der Matrix $R'' \in (\mathbb{Z}/p\mathbb{Z})^{k \times k}$, die entsteht, wenn man aus C die Spalten $1, \ldots, n_1$ und die Zeilen $1, \ldots, n_1 - k$ streicht.				(12)
Setze $d'_p \leftarrow d'_p \cdot d_p \bmod p$ und $d''_p \leftarrow d''_p \cdot d_p \bmod p$.				(13)

In der Schleife (3) - (9) werden die ersten $(n_1 - k)$ Zeilen der Matrix C auf Dreiecksgestalt gebracht. Dies geschieht in derselben Weise wie bei der gewöhnlichen Gauß-Elimination, allerdings werden alle $(n_1 + k)$ Spalten der Matrix bearbeitet.

Dieser Vorgang erfordert für jede bearbeitete Zeile k Spaltenoperationen mehr als die gewöhnliche Gauß-Elimination auf quadratischen Matrizen.

Offensichtlich geht in keine der Spaltenoperationen zur Modifikation einer festen Spalte C_j mit $j \in [\![1, n_1 + k]\!]$ Information aus Spalten C_l mit $l > j$ ein. Ferner geht in keine der Spaltenoperationen zur Modifikation der Spalten $C_{n_1-k+1}, \ldots, C_{n_1+k}$ Information aus anderen als den Spalten $C_1, \ldots, C_{n_1-k}$ ein.

Nach dem Ende der Schleife (3) - (9) haben daher die Spalten $C_1, \ldots, C_{n_1}$ dieselben Einträge, die auch entstanden wären, wenn man die Elimination nur auf der aus diesen Spalten bestehenden quadratischen Matrix C' durchgeführt hätte. Gleiches gilt für die aus den Spalten $C_1, \ldots, C_{n_1-k}, C_{n_1+1}, \ldots, C_{n_1+k}$ gebildete Teilmatrix C'' von C. Auf diese Weise werden also die ersten $(n_1 - k)$ Zeilen beider Teilmatrizen simultan auf Dreiecksgestalt gebracht.

Die verbleibenden jeweils k Zeilen der Matrizen C' und C'' werden anschließend in den Schritten (11) bzw. (12) mit üblicher Gauß-Elimination weiterbehandelt.

Man beachte, daß in Schritt (5) Zeilenvertauschungen durchgeführt werden müssen, wenn ein Diagonalelement den Wert Null hat. Die Durchführung von Spaltenvertauschungen, die bei gewöhnlicher Gauß-Elimination in diesem Fall auch möglich wäre, ist hier nicht zulässig, da die Position der einzelnen Spalten in der Matrix C die Zusammenstellung der beiden Teilmatrizen C' und C'' bestimmt. Es darf also z.B. keine der Spalten $C_{n_1-k+1}, \ldots, C_{n_1+k}$ mit einer der Spalten $C_1, \ldots, C_{n-k}$ vertauscht werden.

Man beachte ferner, daß durchaus $\det B' \equiv \det B'' \equiv 0 \bmod p$ sein kann, obwohl die Matrix B (über $\mathbb{Z}$) als regulär vorausgesetzt wird. Insofern ist Schritt (6) tatsächlich erforderlich.

Die simultane Behandlung der ersten $(n_1 - k)$ Spalten der beiden Teilmatrizen führt zu einer deutlichen Beschleunigung der Determinantenberechnung. Da in unserer Anwendung der Parameter k in den meisten Fällen höchstens den Wert 5 hat, wird hier gegenüber der zweimaligen kompletten Durchführung der Gauß-Elimination fast die Hälfte der Rechenzeit eingespart. Die Berechnung der beiden Determinanten d'_p und d''_p ist also nicht wesentlich aufwendiger als die Berechnung der Determinante von nur einer der beiden Teilmatrizen.

Zur Bestimmung der Determinanten d' und d'' der beiden Teilmatrizen B' und B'' von B muß der Teilalgorithmus GAUSS-SIM für mehrere, geeignet gewählte Primzahlen $p_1, \ldots, p_\nu$ durchgeführt werden. Dabei werden

$$\begin{aligned} d'_\kappa &= \det B' \bmod p_\kappa \quad (1 \le \kappa \le \nu), \\ d''_\kappa &= \det B'' \bmod p_\kappa \quad (1 \le \kappa \le \nu) \end{aligned}$$

bestimmt. Aus diesen Ergebnissen werden anschließend mit dem Chinesischen Restsatz (Algorithmus 3.34) die Determinanten d' und d'' rekonstruiert. Um d' und d'' korrekt ermitteln zu können, müssen die Primzahlen $p_1, \ldots, p_\nu$ so gewählt werden, daß ihr Produkt eine obere Schranke für die Determinanten beider Teilmatrizen ist. Der Teilalgorithmus DETERMINANTE muß dazu wie folgt modifiziert werden:

Algorithmus 10.7 (Teilalgorithmus DETERMINANTE (2. Fassung))

Eingabe: eine Matrix $B \in \mathbb{Z}^{n_1 \times m_1}$ mit $m_1 > n_1$, bei der keine Spalte nur aus Nullen besteht

Ausgabe: falls B regulär ist, ein Vielfaches d von Det $\Lambda(B)$, sonst $d = 0$

Versuche, eine reguläre Teilmatrix $B' \in \mathbb{Z}^{n_1 \times n_1}$ von B auszuwählen. Ordne dabei die Spalten von B so um, daß die Matrix B' aus den Spalten $B_1, \ldots, B_{n_1}$ besteht.	(1)
IF dies ist nicht möglich THEN setze $d \leftarrow 0$ und EXIT.	(2)
Wähle eine zweite reguläre Teilmatrix $B'' \in \mathbb{Z}^{n_1 \times n_1}$ von B aus, die von B' verschieden ist. Ordne dabei die Spalten von B so um, daß für geeignetes $k \in [\![1, m_1 - n_1]\!]$ die Matrix B' aus den Spalten $B_1, \ldots, B_{n_1}$ und die Matrix B'' aus den Spalten $B_1, \ldots, B_{n_1-k}, B_{n_1+1}, \ldots, B_{n+k}$ besteht.	(3)
Bestimme eine obere Schranke $\tilde{d}$ mit $\tilde{d} \geq \max\{\lvert \det B' \rvert, \lvert \det B'' \rvert\}$.	(4)
Bestimme eine Menge $\{p_1, \ldots, p_\nu\}$ von Primzahlen mit $p = \prod_{\kappa=1}^{\nu} p_\kappa > 2\tilde{d}$.	(5)
FOR $\kappa = 1$ TO ν	(6)
Berechne mit dem Algorithmus GAUSS-SIM simultan Zahlen $d'_\kappa, d''_\kappa \in [\![0, p_\kappa - 1]\!]$ mit $d'_\kappa \equiv \det B' \bmod p_\kappa$, $d''_\kappa \equiv \det B'' \bmod p_\kappa$.	(7)
Berechne mit dem Chinesischen Restsatz die eindeutig bestimmte Zahl $d' \in [\![0, p - 1]\!]$, die die Kongruenzen $d' \equiv d'_\kappa \bmod p_\kappa$ $(1 \leq \kappa \leq \nu)$ erfüllt. Falls dann $d' \leq \tilde{d}/2$ ist, setze $d' \leftarrow d' - \tilde{d}$.	(8)
Berechne mit dem Chinesischen Restsatz die eindeutig bestimmte Zahl $d'' \in [\![0, p - 1]\!]$, die die Kongruenzen $d'' \equiv d''_\kappa \bmod p_\kappa$ $(1 \leq \kappa \leq \nu)$ erfüllt. Falls dann $d'' \leq \tilde{d}/2$ ist, setze $d'' \leftarrow d'' - \tilde{d}$.	(9)
Setze $d \leftarrow \gcd(d', d'')$.	(10)

10.4 Auswahl der beiden Teilmatrizen

In diesem Abschnitt erläutern wir unsere Vorgehensweise zur Auswahl der beiden Teilmatrizen B' und B''.

Zur Konstruktion dieser Teilmatrizen ist es notwendig, jeweils n_1 linear unabhängige Spalten der Matrix B zu finden. Die dazu notwendigen Untersuchungen einzelner Spalten von B können - wie später die Berechnung der Determinante - mit den Mitteln der Gauß-Elimination vorgenommen werden.

Daher liegt es nahe, die Konstruktion der Teilmatrizen so in den Vorgang der modularen Gauß-Elimination einzubeziehen, daß dabei bereits die Werte d'_κ und d''_κ für eine oder mehrere der Primzahlen p_κ berechnet werden.

Wir erläutern diese Grundidee anhand des Algorithmus TEILMAT-1, mit dem die Teilmatrix B' ausgewählt wird. Eingabe für diesen Algorithmus ist die Matrix $B \in \mathbb{Z}^{n_1 \times m_1}$ mit $m_1 > n_1$ sowie eine Primzahl p.

Wir führen nun, ähnlich wie im Algorithmus GAUSS-SIM, gewöhnliche Gauß-Elimination modulo p durch und bearbeiten dabei alle m_1 Spalten der Matrix. Falls ein Diagonaleintrag den Wert Null hat, vertauschen wir allerdings in diesem Algorithmus nicht zwei Zeilen der Matrix, sondern zwei Spalten, um einen von Null verschiedenen Eintrag auf die Diagonale zu bringen.

Auf diese Weise werden aus den insgesamt vorhandenen m_1 Spalten n_1 viele so ausgewählt, daß die aus diesen n_1 Spalten bestehende Matrix B' modulo p eine von Null verschiedene Determinante hat. In diesem Fall ist offensichtlich auch $\det B' \neq 0$ über $\mathbb{Z}$. Ferner kann gleichzeitig $d'_p = \det B' \bmod p$ bestimmt werden.

Falls es im Laufe des Algorithmus einmal nicht möglich sein sollte, mit Hilfe von Spaltenvertauschungen einen von Null verschiedenen Eintrag auf die Diagonale zu bringen, so ist $\det B^* \equiv 0 \bmod p$ für jede n_1-reihige Teilmatrix B^* von B. In diesem Fall wird $d'_p = 0$ gesetzt und der Algorithmus beendet.

Ausgabe des Algorithmus ist also zum einen die Matrix B, die eventuell innerhalb des Algorithmus durch Spaltenpermutationen modifiziert wird, und zum anderen eine Zahl $d'_p \in [\![0, p-1]\!]$. Falls $d'_p \neq 0$ ist, ist die aus den ersten n_1 Spalten von B bestehende Teilmatrix B' regulär, und es gilt $d'_p = \det B' \bmod p$.

Algorithmus 10.8 (Teilalgorithmus TEILMAT-1)

Eingabe: B und p wie oben angegeben
Ausgabe: B und d'_p wie oben angegeben

Setze $c_{ij} \leftarrow b_{ij} \bmod p$ für $1 \leq i \leq n_1$, $1 \leq j \leq m_1$ und konstruiere so eine Matrix $C \in (\mathbb{Z}/p\mathbb{Z})^{n_1 \times m_1}$.	(1)
FOR $i = 1$ TO n_1	(2)
IF $c_{ii} = 0$	(3)
THEN Vertausche eine der Spalten $i+1, \ldots, m_1$ von C mit der Zeile i, so daß dann $c_{ii} \neq 0$ ist, und führe dieselbe Spaltenvertauschung auf B durch.	(4)
IF keine solche Spalte existiert THEN setze $d'_p \leftarrow 0$ und EXIT.	(5)
Berechne mit dem Euklidischen Algorithmus $x = (c_{ii})^{-1} \bmod p$.	(6)
FOR $j = i+1$ TO m_1	(7)
Setze (Spalte) $C_j \leftarrow C_j - (x \cdot c_{ij}) \cdot C_i$.	(8)
Setze $d'_p \leftarrow \prod_{i=1}^{n_1} c_{ii} \bmod p$.	(9)

Die Spaltenvertauschungen in Schritt (4) des Algorithmus können in der Praxis durch die Veränderung von Zeigern realisiert werden, erfordern also nicht das Kopieren großer Datenmengen.

Wird der Algorithmus TEILMAT-1 mit der Ausgabe $d'_p = 0$ beendet, so ist, wie bereits erwähnt, $\det B^* \equiv 0 \bmod p$ für jede n_1-reihige Teilmatrix B^* von B. Daraus kann man jedoch nicht schließen, daß B auch über $\mathbb{Z}$ singulär ist. Die Auswahl der Teilmatrix B' kann dann mit der eingegebenen Primzahl p nicht erfolgen; der Algorithmus TEILMAT-1 muß also mit einer anderen Primzahl $\tilde{p}$ erneut durchlaufen werden. Unabhängig davon, wie die Auswahl von B' und B'' später erfolgt, ist aber $d'_p = d''_p = 0$, d.h. die Gauß-Elimination muß für diese Primzahl p später nicht noch einmal wiederholt werden.

Der Algorithmus TEILMAT-1 wird nun solange für verschiedene Primzahlen wiederholt, bis eine von Null verschiedene Determinante gefunden wird und damit B' ausgewählt ist. Falls die Matrix B regulär ist, muß eine Primzahl $\tilde{p}$ existieren, für die dies möglich ist.

Falls B jedoch singulär ist, läßt sich für keine Primzahl $\tilde{p}$ eine reguläre Teilmatrix auswählen. Um in diesem Fall den Versuch der Konstruktion einer regulären Teilmatrix an irgendeiner Stelle abbrechen zu können, benötigen wir Informationen darüber, wieviele erfolglose Durchläufe des Algorithmus TEILMAT-1 notwendig sind, um die Singularität von B nachzuweisen. Diese Informationen erhalten wir mit Hilfe des Chinesischen Restsatzes. Wir bestimmen dazu vor der erstmaligen Durchführung des Algorithmus TEILMAT-1 eine obere Schranke $\tilde{d}$ für die Determinanten aller möglichen n_1-reihigen Teilmatrizen von B sowie Primzahlen $p_1, \ldots, p_\nu$, deren Produkt größer ist als $2\tilde{d}$. Ist nun der Algorithmus TEILMAT-1 für jede dieser Primzahlen erfolglos, ist also $\det B^* \equiv 0 \bmod p_\kappa$ für $1 \leq \kappa \leq \nu$, so folgt mit dem Chinesischen Restsatz, daß auch über $\mathbb{Z}$ die Determinante jeder n_1-reihigen Teilmatrix B^* von B den Wert Null hat. Damit ist die Singularität von B nachgewiesen.

Die Bestimmung der Schranke $\tilde{d}$ erfolgt mit der Hadamardschen Ungleichung (3.3). Wir berechnen dazu zunächst

$$S_j = \left(\sum_{i=1}^{n_1} |b_{ij}|^2 \right)^{1/2} \qquad (1 \leq j \leq m_1).$$

Wir wählen dann unter diesen m_1 vielen Werten S_j die n_1 vielen größten aus und berechnen deren Produkt. Dieses ist offenbar nach (3.3) eine obere Schranke für die Determinante jeder n_1-reihigen Teilmatrix von B.

Wir geben im folgenden die entsprechend modifizierte Fassung des Teilalgorithmus DETERMINANTE an.

Algorithmus 10.9 (Teilalgorithmus DETERMINANTE (3. Fassung))

Eingabe: eine Matrix $B \in \mathbb{Z}^{n_1 \times m_1}$ mit $m_1 > n_1$, bei der keine Spalte nur aus Nullen besteht

Ausgabe: falls B regulär ist, ein Vielfaches d von $\operatorname{Det} \Lambda(B)$, sonst $d = 0$

Bestimme eine obere Schranke $\tilde{d}$, so daß $\tilde{d} > \lvert \det B^* \rvert$ ist für jede beliebige n_1-reihige Teilmatrix B^* von B.		(1)
Bestimme eine Menge $\{p_1, \ldots, p_\nu\}$ von Primzahlen mit $p = \prod_{\kappa=1}^{\nu} p_\kappa > 2\tilde{d}$.		(2)
Setze $\kappa \leftarrow 1$.		(3)
	Durchlaufe Algorithmus TEILMAT-1 mit Eingabe B und p_κ. Die Ausgabe ist B und d'_κ.	(4)
BREAKIF $d'_\kappa \neq 0$.		(5)
	IF $\kappa = \nu$ THEN setze $d \leftarrow 0$ und EXIT.	(6)
	Setze $d''_\kappa \leftarrow 0$.	(7)
	Erhöhe κ um 1.	(8)
/* Die Teilmatrix B' ist nun festgelegt. Die Spalten der Matrix B sind so geordnet, daß die Matrix B' aus den Spalten $B_1, \ldots, B_{n_1}$ besteht. */		
Wähle eine zweite reguläre Teilmatrix $B'' \in \mathbb{Z}^{n_1 \times n_1}$ von B aus, die von B' verschieden ist. Ordne dabei die Spalten von B so um, daß für geeignetes $k \in [\![1, m_1 - n_1]\!]$ die Matrix B' aus den Spalten $B_1, \ldots, B_{n_1}$ und die Matrix B'' aus den Spalten $B_1, \ldots, B_{n_1-k}, B_{n_1+1}, \ldots, B_{n+k}$ besteht. Berechne ferner $d''_\kappa = \det B'' \bmod p_\kappa$.		(9)
WHILE $\kappa \leq \nu$		(10)
	Berechne mit dem Algorithmus GAUSS-SIM simultan Zahlen $d'_\kappa, d''_\kappa \in [\![0, p_\kappa - 1]\!]$ mit $d'_\kappa \equiv \det B' \bmod p_\kappa$, $d''_\kappa \equiv \det B'' \bmod p_\kappa$.	(11)
	Erhöhe κ um 1.	(12)
Berechne mit dem Chinesischen Restsatz die eindeutig bestimmte Zahl $d' \in [\![0, p-1]\!]$, die die Kongruenzen $d' \equiv d'_\kappa \bmod p_\kappa$ $(1 \leq \kappa \leq \nu)$ erfüllt. Falls dann $d' \leq \tilde{d}/2$ ist, setze $d' \leftarrow d' - \tilde{d}$.		(13)
Berechne mit dem Chinesischen Restsatz die eindeutig bestimmte Zahl $d'' \in [\![0, p-1]\!]$, die die Kongruenzen $d'' \equiv d''_\kappa \bmod p_\kappa$ $(1 \leq \kappa \leq \nu)$ erfüllt. Falls dann $d'' \leq \tilde{d}/2$ ist, setze $d'' \leftarrow d'' - \tilde{d}$.		(14)
Setze $d \leftarrow \gcd(d', d'')$.		(15)

Die Auswahl der Teilmatrix B'' wird mit dem Algorithmus TEILMAT-2 vorgenommen, den wir im folgenden erläutern. Dieser Algorithmus arbeitet auf ähnliche Weise wie der Algorithmus TEILMAT-1, berücksichtigt jedoch zusätzlich die Forderung, daß B'' möglichst viele Spalten von B enthalten soll, die nicht bereits zu B' gehören.

Eingabe für den Algorithmus TEILMAT-2 ist die Matrix $B \in \mathbb{Z}^{n_1 \times m_1}$ sowie eine Primzahl p. Die Spalten von B müssen dabei so geordnet sein, daß die bereits festgelegte Teilmatrix B' aus den Spalten $B_1, \ldots, B_{n_1}$ besteht.

Ausgabe ist $d''_p = \det B'' \bmod p$ sowie der Parameter $k \in [\![1, m_1 - n_1]\!]$, der die Anzahl der Spalten angibt, in denen sich B' und B'' unterscheiden. Ferner sind nach Beendigung des Algorithmus die Spalten der Matrix B so umgeordnet, daß die Spalten $B_1, \ldots, B_{n_1}$ zur Teilmatrix B' und die Spalten $B_1, \ldots B_{n_1-k}, B_{n_1+1}, \ldots, B_{n_1+k}$ zur Teilmatrix B'' gehören.

Wir geben im folgenden noch einige grundsätzliche Hinweise zum Ablauf des Algorithmus.

Die Einträge der Matrix B werden zunächst - wie im Algorithmus TEILMAT-1 - in eine Hilfsmatrix C übertragen und dabei modulo p reduziert. Die zu den Spalten der bereits festgelegten Teilmatrix B' von B korrespondierenden Spalten der Matrix C bilden eine Teilmatrix von C, die wir im folgenden mit C' bezeichnen.

Im weiteren Verlauf des Algorithmus wird dann eine zweite Teilmatrix C'' von C ausgewählt, deren Determinante (modulo p) von Null verschieden ist, und zu der möglichst viele Spalten von C gehören, die nicht bereits in C' enthalten sind. Die zu den Spalten aus C'' korrespondierenden Spalten der Matrix B bilden dann später die Matrix B''. Die Auswahl der Teilmatrix C'' geschieht auf dieselbe Weise wie im Algorithmus TEILMAT-1. Die Spalten der Matrix C werden also so vertauscht, daß die aus den ersten n_1 Spalten von C gebildete Teilmatrix modulo p eine von Null verschiedene Determinante hat.

Um zu gewährleisten, daß möglichst viele der nicht zu C' gehörenden Spalten von C für die Teilmatrix C'' ausgewählt werden, müssen diese Spalten in der Matrix C möglichst weit links stehen. Aus diesem Grund werden die Spalten der Matrix C zu Beginn des Algorithmus zunächst so angeordnet, daß die nicht zu C' gehörenden Spalten von C an den Positionen $1, \ldots, (m_1 - n_1)$ stehen. Bei den Spaltenvertauschungen, die während der Auswahl der Teilmatrix C'' notwendig sind, ist ebenfalls darauf zu achten, daß die nicht zu C' gehörenden Spalten von C möglichst weit links bleiben, damit sie bevorzugt für die Matrix C'' ausgewählt werden.

Die auf der Matrix C durchgeführten Spaltenvertauschungen werden - im Gegensatz zu Algorithmus TEILMAT-1 - auf der Matrix B zunächst nicht durchgeführt. Stattdessen wird ein Vektor $s \in \mathbb{Z}^{m_1}$ angelegt, dessen Einträge die Positionen der zu den einzelnen Spalten der Matrix C korrespondierenden Spalten in der Matrix B angeben. Bei jeder Vertauschung von zwei Spalten der Matrix C müssen die entsprechenden Einträge des Vektors s ebenfalls vertauscht werden. Nach der Auswahl der Teilmatrix C'' von C können dann die Spalten von B mit Hilfe der Informationen aus dem Vektor s in der gewünschten Weise permutiert werden.

Nach diesen Vorüberlegungen formulieren wir nun den Algorithmus TEILMAT-2. Einige der Schritte werden anschließend noch genauer erläutert.

Algorithmus 10.10 (Teilalgorithmus TEILMAT-2)

Eingabe: B und p wie oben angegeben
Ausgabe: B, k und d''_p wie oben angegeben

Setze $t \leftarrow m_1 - n_1$.	(1)
Setze $c_{ij} \leftarrow b_{i,n+j} \bmod p$ für $1 \leq i \leq n_1,\ 1 \leq j \leq t$. Setze $c_{ij} \leftarrow b_{i,j-t} \bmod p$ für $1 \leq i \leq n_1,\ t+1 \leq j \leq m_1$. Konstruiere so eine Matrix $C \in (\mathbb{Z}/p\mathbb{Z})^{n_1 \times m_1}$.	(2)
Setze $s_j \leftarrow n_1 + j$ für $1 \leq j \leq t$ und $s_j \leftarrow j - t$ für $t+1 \leq j \leq m_1$.	(3)
FOR $i = 1$ TO n_1	(4)
IF $c_{ii} = 0$	(5)
THEN Es sei j der kleinste Index in $[\![i+1, m_1]\!]$ mit $c_{ij} \neq 0$.	(6)
Vertausche s_i und s_j sowie die Spalten C_i und C_j.	(7)
IF $i \leq t$ und $j > t$	(8)
THEN Erhöhe t um 1.	(9)
Vertausche s_t und s_j sowie die Spalten C_t und C_j.	(10)
Berechne mit dem Euklidischen Algorithmus $x = (c_{ii})^{-1} \bmod p$.	(11)
FOR $j = i+1$ TO m_1	(12)
Setze (Spalte) $C_j \leftarrow C_j - (x \cdot c_{ij}) \cdot C_i$.	(13)
Setze $d''_p \leftarrow \prod_{i=1}^{n_1} c_{ii} \bmod p$.	(14)
Setze $k \leftarrow m_1 - \max\{t, n_1\}$.	(15)
Ordne die Spalten von B so um, daß B' aus den Spalten $B_1, \ldots, B_{n_1}$ und B'' aus den Spalten $B_1, \ldots, B_{n_1-k}, B_{n_1+1}, \ldots, B_{n+k}$ besteht. Passe dabei d''_p entsprechend an und beachte, daß d'_p sich nicht ändern darf. (Teilalgorithmus UMORDNUNG)	(16)

In den Schritten (1) bis (3) wird die Initialisierung des Algorithmus vorgenommen. Gleichzeitig mit der Konstruktion der Matrix C in Schritt (2) werden deren Spalten in der oben angegebenen Weise permutiert. In Schritt (3) wird der Vektor s entsprechend initialisiert. Die Variable t gibt im folgenden jeweils die Nummer der am weitesten rechts stehenden Spalte von C an, die nicht zu C' gehört.

In der Schleife (4) - (13) erfolgt die Auswahl der Teilmatrix C''. Falls in einem der Durchläufe dieser Schleife das Diagonalelement c_{ii} in Schritt (5) den Wert Null hat, muß die Spalte C_i mit einer weiter rechts stehenden Spalte vertauscht werden, so daß anschließend $c_{ii} \neq 0$ ist. In Schritt (6) wird dazu eine möglichst weit links stehende Spalte C_j ausgewählt. Dadurch soll gewährleistet werden, daß die nicht zu C' gehörenden Spalten von C, die zu Beginn der Schleife (4) - (13) ganz links in der Matrix C stehen, möglichst weit links bleiben.

Nun kann es aber notwendig sein, eine der nicht zu C' gehörenden Spalten mit einer zu C' gehörenden Spalte zu vertauschen. In diesem Fall folgt aus der Festlegung von k, daß $i \leq t$ und $j > t$ ist. Die nicht zu C' gehörende Spalte gelangt nun eventuell so weit nach rechts, daß sie bei der Auswahl der Teilmatrix C'' nicht mehr berücksichtigt werden kann. Um dies zu verhindern, wird in den Schritten (9) und (10) eine zweite Spaltenvertauschung durchgeführt. Die nicht zu C' gehörende Spalte, die durch die erste Vertauschung an die Position $j > t$ gelangt ist, wird dabei wieder möglichst weit nach links gebracht, d.h. sie gelangt unmittelbar neben die bisher am weitesten rechts stehende nicht zu C' gehörende Spalte. Der Parameter t wird entsprechend um 1 erhöht.

Man beachte, daß es, wie bereits erwähnt, nicht in jedem Fall möglich ist, alle nicht zu C' gehörenden Spalten von C in die Matrix C'' zu integrieren, da unter diesen Vektoren lineare Abhängigkeiten vorhanden sein können.

Es ist theoretisch sogar denkbar, daß C'' nur Spalten enthält, die auch zu C' gehören. Dieser Fall ist jedoch sehr unwahrscheinlich. Er kann nur auftreten, wenn alle Einträge der nicht zu C' gehörenden Spalten von C Vielfache von p sind.

Die Schritte (11) bis (13) bedürfen keiner weiteren Erläuterung. Sie stimmen mit den entsprechenden Schritten (6) bis (8) des Algorithmus TEILMAT-1 überein. Schritt (14) wird ebenfalls nicht näher erläutert. Er entspricht Schritt (9) des Algorithmus TEILMAT-1.

In Schritt (15) wird anhand des Wertes von t nach Beendigung der Schleife (4) - (13) der Parameter k bestimmt. Falls $t \leq n_1$ ist, konnten alle nicht zu C' gehörenden Spalten von C in die Teilmatrix C'' integriert werden. In diesem Fall ist $k = m_1 - n_1$. Andernfalls konnten einige der nicht zu C' gehörenden Spalten nicht berücksichtigt werden, und es ist $k = m_1 - t$.

In Schritt (16) werden schließlich mit Hilfe des Teilalgorithmus UMORDNUNG die Spalten der Matrix B so umgeordnet, daß die Spalten $B_1, \ldots, B_{n_1}$ zur Teilmatrix B' und die Spalten $B_1, \ldots, B_{n_1-k}, B_{n_1+1}, \ldots, B_{n_1+k}$ zur Teilmatrix B'' gehören. Für diese Umordnung wird neben der ursprünglichen Matrix B der Vektor s benötigt, der im Laufe der Schritte (3) bis (13) des Algorithmus TEILMAT-2 konstruiert wird.

Jede Spalte von B gehört zu einer der folgenden vier Gruppen:

$$\begin{array}{lll}
j \in [\![1, n_1]\!], & s_j \in [\![1, n_1]\!] & \Rightarrow \text{ Spalte } B_{s_j} \text{ gehört zu B' und } B'', \\
j \in [\![1, n_1]\!], & s_j \in [\![n_1+1, m_1]\!] & \Rightarrow \text{ Spalte } B_{s_j} \text{ gehört zu B''}, \\
j \in [\![n_1+1, m_1]\!], & s_j \in [\![1, n_1]\!] & \Rightarrow \text{ Spalte } B_{s_j} \text{ gehört zu B'}, \\
j \in [\![n_1+1, m_1]\!], & s_j \in [\![n_1+1, m_1]\!] & \Rightarrow \text{ Spalte } B_{s_j} \text{ gehört weder zu B'} \\
& & \text{ noch zu } B''.
\end{array}$$

Mit diesen Informationen kann die Matrix B leicht in die gewünschte Form gebracht werden.

Durch die Umordnung der Spalten von B werden implizit auch Spaltenvertauschungen auf den Matrizen B' und B'' durchgeführt, das Vorzeichen der Determinanten dieser beiden Teilmatrizen kann sich also ändern. Daher werden während der Umordnung der Spalten von B zwei Permutationsvektoren r' und r'' berechnet, die

die auf den Matrizen B' bzw. B'' durchgeführten Spaltenvertauschungen angeben. Ist anschließend r' ungerade, so werden die Spalten B_1 und B_2 miteinander vertauscht, so daß die Permutation r' gerade wird, d.h. der bereits mit dem Algorithmus TEILMAT-1 berechnete Wert d'_p muß nicht verändert werden. Ist nun die Permutation r'' ungerade, so wird $d''_p = p - d''_p$ gesetzt.

Algorithmus 10.11 (Teilalgorithmus UMORDNUNG)

Eingabe: B, n_1, m_1, k, s, d''_p wie in Algorithmus TEILMAT-2 vor Schritt (16)
Ausgabe: B und d''_p wie oben angegeben

Setze $B^* \leftarrow B$.			(1)
Setze $i_1 \leftarrow 0$, $i_2 \leftarrow n_1 - k$, $i_3 \leftarrow n_1$ und $i_4 \leftarrow n_1 + k$.			(2)
FOR $j = 1$ TO n_1			(3)
	IF $s_j < n_1$		(4)
	THEN	Erhöhe i_1 um 1 und setze dann $B_{i_1} \leftarrow B^*_{s_j}$.	(5)
		Setze $r'_{i_1} \leftarrow s_j$ und $r''_{i_1} \leftarrow j$.	(6)
	ELSE	Erhöhe i_3 um 1 und setze dann $B_{i_3} \leftarrow B^*_{s_j}$.	(7)
		Setze $r''_{i_3-k} \leftarrow j$.	(8)
FOR $j = n_1 + 1$ TO m_1			(9)
	IF $s_j < n_1$		(10)
	THEN	Erhöhe i_2 um 1 und setze dann $B_{i_2} \leftarrow B^*_{s_j}$.	(11)
		Setze $r'_{i_1} \leftarrow s_j$.	(12)
	ELSE	Erhöhe i_4 um 1 und setze dann $B_{i_4} \leftarrow B^*_{s_j}$.	(13)
IF die Permutation r' ist ungerade THEN vertausche r''_1 und r''_2 sowie die Spalten B_1 und B_2.			(14)
IF die Permutation r'' ist ungerade THEN setze $d''_p \leftarrow p - d''_p$.			(15)

Wir geben nun die endgültige Fassung des Teilalgorithmus DETERMINANTE an.

Man beachte, daß die Schleife (9) - (12) nur dann verlassen wird, wenn $k > 0$ ist. Auf diese Weise wird erzwungen, daß sich die beiden Teilmatrizen B' und B'' in mindestens einer Spalte unterscheiden. Der Fall $k = 0$ ist jedoch in der Praxis nie aufgetreten.

Ferner sei erwähnt, daß in der Praxis in fast allen Fällen auch die Schleife (4) - (8) nur einmal durchlaufen wurde. Die beiden Teilmatrizen konnten also in der Regel mit der ersten getesteten Primzahl ausgewählt werden.

Algorithmus 10.12 (Teilalgorithmus DETERMINANTE (Endfassung))

Eingabe: eine Matrix $B \in \mathbb{Z}^{n_1 \times m_1}$ mit $m_1 > n_1$, bei der keine Spalte nur aus Nullen besteht

Ausgabe: falls B regulär ist, ein Vielfaches d von $\operatorname{Det} \Lambda(B)$, sonst $d = 0$

Bestimme eine obere Schranke $\tilde{d}$, so daß $\tilde{d} > \lvert \det B^* \rvert$ ist für jede beliebige n_1-reihige Teilmatrix B^* von B.		(1)
Bestimme eine Menge $\{p_1, \ldots, p_\nu\}$ von Primzahlen mit $p = \prod_{\kappa=1}^{\nu} p_\kappa > 2\tilde{d}$.		(2)
Setze $\kappa \leftarrow 1$.		(3)
	Durchlaufe Algorithmus TEILMAT-1 mit Eingabe B und p_κ. Die Ausgabe ist B und d'_κ.	(4)
BREAKIF $d'_\kappa \neq 0$.		(5)
	IF $\kappa = \nu$ THEN setze $d \leftarrow 0$ und EXIT.	(6)
	Setze $d''_\kappa \leftarrow 0$.	(7)
	Erhöhe κ um 1.	(8)
/* Die Teilmatrix B' ist nun festgelegt. Die Spalten der Matrix B sind so geordnet, daß die Matrix B' aus den Spalten $B_1, \ldots, B_{n_1}$ besteht. */		
	Durchlaufe Algorithmus TEILMAT-2 mit Eingabe B und p_κ. Die Ausgabe ist B, d''_κ und k.	(9)
BREAKIF $k > 0$		(10)
	Erhöhe κ um 1.	(11)
	Bestimme mit gewöhnlicher Gauß-Elimination $d'_\kappa = \det B' \bmod p$.	(12)
/* Die Teilmatrix B'' ist nun festgelegt. Die Spalten der Matrix B sind so geordnet, daß die Matrix B' aus den Spalten $B_1, \ldots, B_{n_1}$ und die Matrix B'' aus den Spalten $B_1, \ldots, B_{n_1-k}, B_{n_1+1}, \ldots, B_{n_1+k}$ besteht. */		
WHILE $\kappa \leq \nu$		(13)
	Berechne mit dem Algorithmus GAUSS-SIM simultan Zahlen $d'_\kappa, d''_\kappa \in [\![0, p_\kappa - 1]\!]$ mit $d'_\kappa \equiv \det B' \bmod p_\kappa$, $d''_\kappa \equiv \det B'' \bmod p_\kappa$.	(14)
	Erhöhe κ um 1.	(15)
Berechne mit dem Chinesischen Restsatz die eindeutig bestimmte Zahl $d' \in [\![0, p-1]\!]$, die die Kongruenzen $d' \equiv d'_\kappa \bmod p_\kappa$ $(1 \leq \kappa \leq \nu)$ erfüllt. Falls dann $d' \leq \tilde{d}/2$ ist, setze $d' \leftarrow d' - \tilde{d}$.		(16)
Berechne mit dem Chinesischen Restsatz die eindeutig bestimmte Zahl $d'' \in [\![0, p-1]\!]$, die die Kongruenzen $d'' \equiv d''_\kappa \bmod p_\kappa$ $(1 \leq \kappa \leq \nu)$ erfüllt. Falls dann $d'' \leq \tilde{d}/2$ ist, setze $d'' \leftarrow d'' - \tilde{d}$.		(17)
Setze $d \leftarrow \gcd(d', d'')$.		(18)

Wir untersuchen nun noch, wie die Auswahl der Primzahlen $p_1, \ldots, p_\nu$ in Schritt (2) des Algorithmus DETERMINANTE erfolgen muß, damit der Algorithmus in der Praxis effizient ist. Dabei sind zwei Gesichtspunkte zu beachten.

Einerseits soll die Anzahl ν der Primzahlen möglichst klein sein, damit nur wenige Durchläufe der Algorithmen TEILMAT-1, TEILMAT-2 und GAUSS-SIM notwendig sind. Da das Produkt der Primzahlen eine vorgegebene Schranke übersteigen muß, kann dieses Ziel nur erreicht werden, wenn die einzelnen Primzahlen relativ groß sind.

Andererseits soll die Arithmetik in den Algorithmen TEILMAT-1, TEILMAT-2 und GAUSS-SIM effizient durchführbar sein. Dazu ist es in der Praxis erforderlich, daß die einzelnen Primzahlen unterhalb von 2^{32} liegen, damit alle arithmetischen Operationen mit der üblichen 32-Bit-Arithmetik durchgeführt werden können, die auf den verwendeten Rechnern zur Verfügung steht.

Wir wählen daher die Primzahlen $p_1, \ldots, p_\nu$ aus einer Liste, die die Primzahlen unterhalb von 2^{32} in absteigender Reihenfolge enthält.

Die folgende Tabelle gibt für die einzelnen Diskriminanten die Anzahl ν der Primzahlen an, für die die modulare Gauß-Elimination durchgeführt werden muß. Zusätzlich geben wir die Größenordnung $\overline{\tilde{d}}$ der in Schritt (1) des Algorithmus DETERMINANTE berechneten oberen Schranke $\tilde{d}$ an. Da für steigende Diskriminante auch die Größe der Matrix B wächst (vgl. dazu die Tabelle in Kapitel 9), steigt auch die in Schritt (1) des Algorithmus DETERMINANTE berechnete obere Abschätzung $\tilde{d}$ und damit die Anzahl ν der benötigten Primzahlen.

Tabelle 10.13 (Anzahl der benötigten Primzahlen)

$\lvert\Delta\rvert$	$\overline{\tilde{d}}$	ν
$4 \cdot 10^{10} + 4$	10^{18}	2
$4 \cdot 10^{11} + 4$	10^{18}	2
$4 \cdot 10^{12} + 4$	10^{18}	2
$4 \cdot 10^{13} + 4$	10^{26}	3
$4 \cdot 10^{14} + 4$	10^{23}	3
$4 \cdot 10^{15} + 4$	10^{33}	4
$4 \cdot 10^{16} + 4$	10^{38}	4
$4 \cdot 10^{17} + 4$	10^{52}	6
$4 \cdot 10^{18} + 4$	10^{56}	6
$4 \cdot 10^{19} + 4$	10^{62}	7
$4 \cdot 10^{20} + 4$	10^{72}	8
$4 \cdot 10^{21} + 4$	10^{89}	10
$4 \cdot 10^{22} + 4$	10^{94}	10
$4 \cdot 10^{23} + 4$	10^{112}	12
$4 \cdot 10^{24} + 4$	10^{150}	16
$4 \cdot 10^{25} + 4$	10^{195}	21
$4 \cdot 10^{26} + 4$	10^{201}	21
$4 \cdot 10^{27} + 4$	10^{231}	24

$\lvert\Delta\rvert$	$\overline{\tilde{d}}$	ν
$4 \cdot 10^{28} + 4$	10^{221}	23
$4 \cdot 10^{29} + 4$	10^{271}	29
$4 \cdot 10^{30} + 4$	10^{290}	31
$4 \cdot 10^{31} + 4$	10^{310}	33
$4 \cdot 10^{32} + 4$	10^{365}	38
$4 \cdot 10^{33} + 4$	10^{489}	51
$4 \cdot 10^{34} + 4$	10^{441}	46
$4 \cdot 10^{35} + 4$	10^{478}	50
$4 \cdot 10^{36} + 4$	10^{565}	59
$4 \cdot 10^{37} + 4$	10^{648}	68
$4 \cdot 10^{38} + 4$	10^{621}	65
$4 \cdot 10^{39} + 4$	10^{689}	72
$4 \cdot 10^{40} + 4$	10^{642}	67
$4 \cdot 10^{41} + 4$	10^{864}	90
$4 \cdot 10^{42} + 4$	10^{826}	86
$4 \cdot 10^{43} + 4$	10^{879}	92
$4 \cdot 10^{44} + 4$	10^{984}	103
$4 \cdot 10^{45} + 4$	10^{1077}	112

10.5 Die Generierung zusätzlicher Relationen

Die Generierung zusätzlicher Relationen kann im Verlauf des Rahmenalgorithmus MODUL an drei Stellen, nämlich in den Schritten (3), (7) und (11), notwendig sein. In allen diesen Fällen müssen Relationen über der in Schritt (4) des Algorithmus CLASSGROUP berechneten verkleinerten Formenbasis F_1 generiert werden.

Nun wurde bereits in Kapitel 9 erläutert, daß es mit der in Kapitel 8 beschriebenen Vorgehensweise nicht direkt möglich ist, Relationen über der Formenbasis F_1 zu generieren. Es ist daher zum Auffinden einer neuen Relation über F_1 notwendig, zunächst eine Relation über der ursprünglichen Formenbasis F zu generieren und diese dann mit Algorithmus 9.2 in eine Relation über F_1 umzurechnen.

Wir formulieren dazu im folgenden den Algorithmus REL-ZUSATZ. Dieser Algorithmus stimmt im wesentlichen mit dem in Abschnitt 8.5 angegebenen Algorithmus RELATIONEN überein, beinhaltet jedoch zusätzlich die oben erwähnte Umrechnung der Relationen. Ferner entfallen alle Schritte, die im Zusammenhang mit dem Ziel der Konstruktion einer regulären Matrix stehen.

Eingabe für den Algorithmus REL-ZUSATZ sind zunächst alle Daten, die auch der Algorithmus RELATIONEN als Eingabe erhält. Diese Daten werden für die Generierung der Relationen über der ursprünglichen Formenbasis F benötigt. Im einzelnen handelt es sich dabei um die Diskriminante Δ, die ursprüngliche Formenbasis F, die Menge F_p der zu den Primformen aus F gehörigen Primzahlen, die Anzahl n der Elemente in F, die Anzahl k der gewöhnlichen Primformen in F sowie die Menge M der mehrfachen Primteiler der Diskriminante unterhalb von $6 \log^2 |\Delta|$.

Ferner erhält der Algorithmus die Anzahl k_1 der Elemente in der verkleinerten Formenbasis F_1, die Umformungsmatrix C und den entsprechenden Umformungsvektor u als Eingabe. Diese Daten werden im Verlauf von Schritt (4) des Algorithmus CLASSGROUP berechnet und sind für die Umrechnung der Relationen notwendig.

Schließlich ist noch die Anzahl a der zu berechnenden Relationen anzugeben.

Die während des Algorithmus REL-ZUSATZ verwendeten Teilalgorithmen INITPOTENZ, POTENZPROD, FAKTOR, RELVEKTOR und LARGEREL können unverändert aus Kapitel 8 übernommen werden.

Algorithmus 10.14 (REL-ZUSATZ)

Eingabe: Δ, F, F_p, n, k und M wie in Kapitel 8,
k_1, C und u wie in Kapitel 9),
die Anzahl a der zu generierenden Relationen
Ausgabe: a Relationen $z_1, \ldots, z_a \in \mathbb{Z}^{k_1}$ über der Formenbasis F_1

- Setze $k_0 \leftarrow \max\Big(\{1 \le i \le k_1 \;:\; p_i < \log|\Delta|\} \cup \{4\}\Big)$, $t \leftarrow \max\{4, \lfloor \frac{k_0}{2} \rfloor\}$ und $B \leftarrow 30$. (1)
- Berechne die reduzierten Formen h_{ij} in den Klassen $[f_i^j]$ für $1 \le i \le k_0$, $1 \le j \le B$. (Teilalgorithmus INITPOTENZ) (2)
- Setze $k_1 \leftarrow \lfloor \frac{n}{4} \rfloor$.
  Setze $k_2 \leftarrow \max\{\lfloor \frac{n}{2} \rfloor, n'\}$, wobei $n' = \min\{1 \le i \le k \;: p_i^2 > p_k\}$ ist. (3)
- Setze $m \leftarrow 1$ und $L \leftarrow p_{k_2}^2$. (4)
- WHILE $m \le a$ (5)
  - REPEAT
    - Wähle einen Exponentenvektor $x = (x_1, \ldots, x_n) \in \mathbb{Z}^n$ und berechne die reduzierte Form $g = (a, b)$ in der Klasse von $\prod_{i=1}^{n} f_i^{x_i}$. (Teilalgorithmus POTENZPROD) (6)
    - Stelle fest, ob a vollständig über F_p zerfällt und, falls ja, berechne den Zerlegungsvektor $y = (y_1, \ldots, y_n) \in \mathbb{Z}^n$ mit $a = \prod_{i=1}^{n} p_i^{y_i}$. Gib ferner den unzerlegbaren Anteil r von a an. (Teilalgorithmus FAKTOR) (7)
  - UNTIL $r \le L$. (8)
  - Berechne den (vollständigen oder unvollständigen) Relationenvektor $w = (w_1, \ldots, w_n) \in \mathbb{Z}^n$. (Teilalgorithmus RELVEKTOR) (9)
  - IF $r > 1$ (10)
  - THEN
    - IF in der Liste der unvollständigen Relationen existiert ein Eintrag (r', w', b') mit $r' = r$ (11)
    - THEN
      - Berechne aus (r, w, b) und (r', w', b') eine vollständige Relation $v \in \mathbb{Z}^n$. (Teilalgorithmus LARGEREL) (12)
      - Setze $w \leftarrow v$ und $r \leftarrow 1$. (13)
    - ELSE
      - Speichere (r, w, b) in der Liste der unvollständigen Relationen. (14)
  - IF $r = 1$ und w ist nicht der Nullvektor (15)
  - THEN
    - Berechne mit Hilfe von Algorithmus 9.2 aus der Relation w über F die Relation z_m über F_1. (16)
    - IF z_m ist nicht der Nullvektor THEN setze $m \leftarrow m + 1$. (17)

Kapitel 11

Abschließende Berechnungen

In diesem Kapitel beschreiben wir zunächst die Schritte (8) bis (13) des Algorithmus CLASSGROUP. Wir geben im ersten Abschnitt einen kurzen Überblick über den Ablauf dieser Schritte. Im zweiten und dritten Abschnitt gehen wir genauer auf die Schritte (12) und (13) ein.

Im vierten Abschnitt geben wir dann die Klassenzahl und Struktur der Klassengruppe für die in den bisherigen Tabellen als Beispiele verwendeten Diskriminanten sowie die Rechenzeiten zur Ermittlung dieser Ergebnisse an.

11.1 Überblick über die Schritte (8) bis (13)

Im folgenden bezeichne Γ das volle Relationengitter über der in Schritt (4) berechneten verkleinerten Formenbasis F_1. Die Anzahl der Elemente in der Formenbasis F_1 beträgt n_1, also ist Γ ein n_1-dimensionales Gitter im $\mathbb{Z}^{n_1}$.

Nach Beendigung der Schleife (5) - (7) des Algorithmus CLASSGROUP liegt eine reguläre Matrix $B \in \mathbb{Z}^{n_1 \times m_1}$ mit $m_1 > n_1$ vor, deren Spalten ein n_1-dimensionales Teilgitter von Γ erzeugen. Ferner ist ein Vielfaches d der Determinante dieses Teilgitters bekannt.

In Schritt (8) wird nun mit modularer Hermite-Reduktion eine Basis H des Gitters $\Lambda(B)$ in Hermite-Normalform berechnet, wobei d als Modul verwendet wird. Die Hermite-Reduktion wird mit dem in Kapitel 3 dargestellten Algorithmus 3.22 durchgeführt. Wir gehen darauf hier nicht mehr genauer ein.

In Schritt (9) wird die Determinante h der Matrix H berechnet. Da die Matrix H in Hermite-Normalform vorliegt, sind dazu lediglich ihre Diagonaleinträge zu multiplizieren. In Schritt (10) wird dann diese Determinante h mit der Näherung $2h^*$ verglichen. Ist $h < 2h^*$, so erzeugen die Spalten der Matrix H das volle Relationengitter Γ, d.h. h ist die gesuchte Klassenzahl. In diesem Fall wird der Algorithmus CLASSGROUP in Schritt (13) fortgesetzt.

Falls jedoch $h \geq 2h^*$ ist, so erzeugen die Spalten der Matrix H ein echtes Teilgitter von Γ, das Gitter $\Lambda(H)$ muß also noch durch geeignete zusätzliche Relationen erweitert werden. Dazu werden die Schritte (11) und (12) durchlaufen.

In Schritt (11) wird zunächst eine zusätzliche Relation v über der Formenbasis F_1 generiert. Dies geschieht in derselben Weise wie in Schritt (7), d.h. mit dem in Abschnitt 10.5 dargestellten Algorithmus REL-ZUSATZ. Der Eingabeparameter k erhält in diesem Fall den Wert 1.

In Schritt (12) wird anschließend eine Basis des durch die neue Relation v erweiterten Gitters $\Lambda(H)$ in Hermite-Normalform berechnet. Dazu ist es nicht notwendig, erneut eine vollständige Hermite-Reduktion durchzuführen, da die Matrix H bereits in Hermite-Normalform vorliegt. Wir bezeichnen diesen Vorgang als die **Wiederherstellung der Hermite-Normalform**. Den entsprechenden Algorithmus geben wir in Abschnitt 11.2 an.

Nach der Durchführung der Schritte (11) und (12) werden wiederum die Schritte (9) und (10) durchlaufen, um festzustellen, ob die Spalten der Matrix H nun das volle Relationengitter Γ erzeugen. Vor der Durchführung von Schritt (13) ist somit, eventuell nach mehrfachem Durchlaufen der Schleife (9) - (12), eine Basis H des vollen Relationengitters Γ in Hermite-Normalform sowie die Klassenzahl $h(\Delta)$ bekannt.

In Schritt (13) wird schließlich die Struktur und ein minimales Erzeugendensystem der Klassengruppe $Cl(\Delta)$ berechnet. Dazu muß der in Kapitel 3 angegebene Algorithmus zur Smith-Reduktion geringfügig modifiziert werden. Wir gehen hierauf in Abschnitt 11.3 genauer ein.

Die Generierung zusätzlicher Relationen in der Schleife (9) - (12) des Algorithmus CLASSGROUP ist in der Regel in etwa der Hälfte aller Fälle erforderlich. Dabei ist meistens eine einzige neue Relation ausreichend, um das in Schritt (8) vorliegende Teilgitter Γ' zum vollen Relationengitter Γ zu erweitern.

Die folgende Tabelle enthält dazu einige Daten. Für die einzelnen Diskriminanten wird die Anzahl a der Durchläufe der Schritte (11) und (12) angegeben, die für die Erweiterung des Teilgitters Γ' zum vollen Relationengitter Γ notwendig waren. Ferner wird der Faktor D^* angegeben, um den sich die Determinante von Γ', also der im ersten Durchlauf von Schritt (9) berechnete Wert für h, noch von der Klassenzahl $h(\Delta)$ unterscheidet.

Tabelle 11.1 (Zusätzliche Relationen in der Schleife (9) - (12))

$\|\Delta\|$	a	D^*	$\|\Delta\|$	a	D^*	$\|\Delta\|$	a	D^*
$4 \cdot 10^{10} + 4$	1	2	$4 \cdot 10^{22} + 4$	1	2	$4 \cdot 10^{34} + 4$	–	–
$4 \cdot 10^{11} + 4$	1	2	$4 \cdot 10^{23} + 4$	–	–	$4 \cdot 10^{35} + 4$	3	4
$4 \cdot 10^{12} + 4$	–	–	$4 \cdot 10^{24} + 4$	1	2	$4 \cdot 10^{36} + 4$	–	–
$4 \cdot 10^{13} + 4$	–	–	$4 \cdot 10^{25} + 4$	–	–	$4 \cdot 10^{37} + 4$	1	2
$4 \cdot 10^{14} + 4$	1	2	$4 \cdot 10^{26} + 4$	–	–	$4 \cdot 10^{38} + 4$	2	2
$4 \cdot 10^{15} + 4$	–	–	$4 \cdot 10^{27} + 4$	2	2	$4 \cdot 10^{39} + 4$	–	–
$4 \cdot 10^{16} + 4$	4	4	$4 \cdot 10^{28} + 4$	1	2	$4 \cdot 10^{40} + 4$	2	2
$4 \cdot 10^{17} + 4$	–	–	$4 \cdot 10^{29} + 4$	–	–	$4 \cdot 10^{41} + 4$	–	–
$4 \cdot 10^{18} + 4$	1	4	$4 \cdot 10^{30} + 4$	–	–	$4 \cdot 10^{42} + 4$	–	–
$4 \cdot 10^{19} + 4$	–	–	$4 \cdot 10^{31} + 4$	1	2	$4 \cdot 10^{43} + 4$	1	2
$4 \cdot 10^{20} + 4$	–	–	$4 \cdot 10^{32} + 4$	2	27	$4 \cdot 10^{44} + 4$	–	–
$4 \cdot 10^{21} + 4$	–	–	$4 \cdot 10^{33} + 4$	–	–	$4 \cdot 10^{45} + 4$	–	–

11.2 Wiederherstellung der Hermite-Normalform

In diesem Abschnitt erläutern wir Schritt (12) des Algorithmus CLASSGROUP. Eingabe ist eine Matrix $H \in \mathbb{Z}^{n_1 \times n_1}$ in Hermite-Normalform und ein Vektor $v \in \mathbb{Z}^{n_1}$. Aus Schritt (9) ist ferner $h = \det H$ bekannt. Das Ziel besteht darin, eine Basis des von den Spalten der Matrix H und dem Vektor v erzeugten Gitters zu berechnen, die wiederum in Hermite-Normalform vorliegen soll.

Formal ist also Hermite-Reduktion auf der Matrix $H' \in \mathbb{Z}^{n_1 \times (n_1+1)}$ mit

$$\begin{aligned} H'_i &= H_i \qquad (1 \leq i \leq n_1), \\ H'_{n_1+1} &= v \end{aligned}$$

durchzuführen. Da $h = \det H = \operatorname{Det} \Lambda(H)$ ein Vielfaches der Determinante des erweiterten Gitters $\Lambda(H')$ ist, kann die Hermite-Reduktion modular durchgeführt werden. Da ferner die Matrix H bereits in Hermite-Normalform vorliegt, können viele Schritte der üblichen modularen Hermite-Reduktion entfallen. Wir geben im folgenden den entsprechenden Algorithmus an.

Algorithmus 11.2 (Wiederherstellung der Hermite-Normalform)

Eingabe: eine Matrix $H' \in \mathbb{Z}^{n_1 \times n_1}$ in Hermite-Normalform und ein Vektor $v \in \mathbb{Z}^{n_1}$ sowie $d = \det H'$

Ausgabe: eine Basis H des von den Spalten der Matrix H' sowie dem Vektor v erzeugten Gitters in Hermite-Normalform

Setze $j \leftarrow n_1 + 1$ und (Spalte) $H'_j \leftarrow v$.	(1)
/* Bestimme eine Basis H des Gitters $\Lambda(H')$ in unterer Dreiecksform mit positiven Diagonaleinträgen. */	
Setze $H \leftarrow H'$.	(2)
FOR $i = 1$ TO n_1	(3)
Berechne mit dem Euklidischen Algorithmus $r = \gcd(h_{ii}, h_{ij})$ und die Koeffizienten p, q aus der Darstellung $r = ph_{ii} + qh_{ij}$.	(4)
Setze (Spalte) $H_i \leftarrow pH_i + qH_j$. Setze (Vektor) $H_j \leftarrow \frac{h_{ii}}{r} H_j - \frac{h_{ij}}{r} H_i$.	(5)
Reduziere alle neu berechneten Einträge in den Spalten H_i und H_j modulo d.	(6)
Entferne die Spalte H_j aus H.	(7)
/* Reduziere die Einträge unterhalb der Diagonalen. */	
FOR $i = 2$ TO n	(8)
Führe mit Algorithmus 3.18 elementare Spaltenoperationen auf den Spalten 1 bis i von H durch, so daß $0 \leq h_{ij} < h_{ii}$ für $1 \leq j < i$. Reduziere dabei bei jeder Spaltenoperation die Einträge an den Positionen $k = i+1, \ldots, n$ der veränderten Spalten modulo d.	(9)

Die in jedem Durchlauf von Schritt (4) durchgeführten Transformationen der Matrix H sind auf dieselbe Weise konstruiert wie in Schritt (5) von Algorithmus 3.17. Es handelt sich also, wie sich aus der Argumentation in Abschnitt 3.2 ergibt, um unimodulare Spaltentransformationen. Auf diese Weise werden die Einträge der Spalte H_j sukzessive von oben nach unten zu Null gemacht. Die Dreiecksgestalt der ersten n_1 Spalten der Matrix H bleibt dabei erhalten. Nach dem Ende der Schleife (3) - (5) hat also die Matrix H - wie gewünscht - Dreiecksform mit positiven Diagonalelementen.

Man beachte, daß im Algorithmus 11.2 gegenüber der üblichen Hermite-Reduktion wesentlich weniger Spaltenoperationen benötigt werden, um die Matrix H in Dreiecksgestalt zu überführen. Damit ist ein beträchtlicher Laufzeitgewinn verbunden.

Wir untersuchen nun die in Schritt (5) durchgeführte Reduktion der neu berechneten Einträge modulo d. Hier werden geeignete Vielfache der Vektoren

$$dE_k \qquad (i+1 \leq k \leq n_1)$$

zu den Vektoren H_i und H_j addiert. Offensichtlich sind die Vektoren dE_k Linearkombinationen der Spalten $H_k, \ldots, H_{n_1}$, d.h. in Schritt (5) wird zu den Spalten H_i bzw. H_j ein Vielfaches der Spalten $H_{i+1}, \ldots, H_{n_1}$ addiert. Die in Schritt (5) durchgeführten Veränderungen sind also ebenfalls unimodulare Spaltentransformationen. Nach Beendigung der Schleife (3) - (5) ist also die Matrix H tatsächlich eine Basis des Gitters $\Lambda(H')$.

Es ist also hier, im Gegensatz zur gewöhnlichen modularen Hermite-Reduktion, nicht erforderlich, nach der Überführung der Matrix H in Dreiecksform eine Basis des ursprünglichen Gitters zu rekonstruieren. Die Schritte (5) bis (9) der üblichen modularen Hermite-Reduktion können somit hier entfallen.

Die übrigen Schritte des Algorithmus 11.2 bedürfen keiner weiteren Erläuterung. Ihre Korrektheit ergibt sich unmittelbar aus der Argumentation in Kapitel 3.

11.3 Die Durchführung der Smith-Reduktion

In diesem Abschnitt beschreiben wir Schritt (13) des Algorithmus CLASSGROUP. Hier werden Struktur und minimales Erzeugendensystem der Klassengruppe berechnet. Eingabe für diesen letzten Schritt des Algorithmus CLASSGROUP ist die Matrix $H \in \mathbb{Z}^{n_1 \times n_1}$ in Hermite-Normalform, deren Spalten eine Basis für das volle Relationengitter Γ über der Formenbasis F_1 sind.

Zur Bestimmung der Struktur der Klassengruppe ist es notwendig, die Smith-Normalform S der Matrix H zu berechnen. Ist dann $k \in [\![1, n_1]\!]$ der kleinste Index mit $s_{ii} > 1$, so ist $(s_k, \ldots, s_{n_1})$ die Struktur der Klassengruppe.

Die grundsätzliche Idee zur gleichzeitigen Berechnung eines minimalen Erzeugendensystems wurde bereits in Abschnitt 6.3 angegeben. Sie besteht darin, während der Smith-Reduktion der Matrix H die zu der linksseitigen Transformationsmatrix

U inverse Matrix X zu bestimmen. Es ist dann $([X_k], \dots, [X_{n_1}])$ ein minimales Erzeugendensystem für die Gruppe $G = \mathbb{Z}^{n_1}/\Gamma$ und daher

$$[g_j] = \prod_{i=1}^{n_1} [f_{r_i}]^{x_{ij}} \qquad (k \leq j \leq n_1)$$

ein minimales Erzeugendensystem für die Klassengruppe. Hierbei ist $r \in \mathbb{Z}^{n_1}$ der in Schritt (4) des Algorithmus CLASSGROUP berechnete Positionenvektor, der die Nummern der einzelnen Elemente der verkleinerten Formenbasis F_1 in der ursprünglichen Formenbasis F angibt.

Nun zeigt die praktische Erfahrung, daß die meisten Diagonalelemente der Matrix H bereits vor Beginn von Schritt (13) den Wert 1 haben. Es bietet sich daher an, vor Beginn der Smith-Reduktion eine weitere Verkleinerung des Relationensystems gemäß Satz 6.6 durchzuführen. Bei dieser Verkleinerung werden alle Zeilen aus H eliminiert, bei denen der Diagonaleintrag den Wert 1 hat. Da die Matrix H Hermite-Normalform hat, haben alle weiteren Einträge dieser Zeilen den Wert Null, so daß die Verkleinerung des Relationensystems ohne Spaltenoperationen erfolgen kann. Diese Verkleinerung des Relationensystems hat zur Folge, daß die Smith-Reduktion nur noch auf eine sehr kleine Matrix angewandt werden muß.

Mit diesen Überlegungen ergibt sich der folgende Algorithmus zur Realisierung von Schritt (13) des Algorithmus CLASSGROUP.

Algorithmus 11.3 (Struktur und minimales Erzeugendensystem)

Eingabe: die Matrix $H \in \mathbb{Z}^{n_1 \times n_1}$ und der Positionenvektor $r \in \mathbb{Z}^{n_1}$
Ausgabe: Struktur und minimales Erzeugendensystem der Klassengruppe

Setze $S \leftarrow H$ und $n_2 \leftarrow n_1$.	(1)
WHILE es existiert ein Index $i \in \llbracket 1, n_2 \rrbracket$ mit $s_{ii} = 1$	(2)
Streiche Zeile i und Spalte i aus der Matrix S.	(3)
Streiche den i-ten Eintrag aus dem Positionenvektor r.	(4)
Vermindere n_2 um 1.	(5)
Berechne mit Algorithmus 3.28 die Diagonaleinträge $s_1, \dots, s_{n_2}$ der Smith-Normalform von G. Bestimme dabei auch die zur linksseitigen Transformationsmatrix U inverse Matrix X.	(6)
Es sei k der kleinste Index in $\llbracket 1, n_2 \rrbracket$ mit $s_i > 1$.	(7)
FOR $j = k$ TO n_2	(8)
Gib s_j als $(j-k+1)$-ten Eintrag der Struktur der Klassengruppe aus.	(9)
Gib die Klasse $[g_j] = \prod_{i=1}^{n_1} [f_{r_i}]^{x_{ij}}$ als $(j-k+1)$-tes Element des minimalen Erzeugendensystems der Klassengruppe aus.	(10)

In der folgenden Tabelle geben wir die Größe n_1 der Matrix S bei Eintritt in den Algorithmus 11.3 und die Größe n_2 der Matrix S nach Durchführung der Verkleinerung in den Schritten (2) bis (5) dieses Algorithmus an. Die angegebenen Daten zeigen, daß auf diese Weise die Größe der mit Smith-Reduktion zu behandelnden Matrix tatsächlich beachtlich verkleinert werden kann. Wir geben ferner die Größe n_3 des minimalen Erzeugendensystems an. Man beachte, daß n_3 in den meisten Fällen nicht wesentlich kleiner ist als n_2. Um einen direkten Vergleich der unterschiedlichen Größenordnungen zu ermöglichen, geben wir zusätzlich die Größe n der ursprünglichen Formenbasis F an.

Tabelle 11.4 (Weitere Verkleinerung der Relationenmatrix)

$\|\Delta\|$	n	n_1	n_2	n_3
$4 \cdot 10^{10} + 4$	249	9	4	3
$4 \cdot 10^{11} + 4$	300	10	5	4
$4 \cdot 10^{12} + 4$	339	7	4	3
$4 \cdot 10^{13} + 4$	385	11	4	3
$4 \cdot 10^{14} + 4$	459	10	5	4
$4 \cdot 10^{15} + 4$	483	13	8	7
$4 \cdot 10^{16} + 4$	556	17	9	5
$4 \cdot 10^{17} + 4$	613	20	4	4
$4 \cdot 10^{18} + 4$	649	22	6	3
$4 \cdot 10^{19} + 4$	708	24	3	2
$4 \cdot 10^{20} + 4$	796	27	5	4
$4 \cdot 10^{21} + 4$	896	33	8	7
$4 \cdot 10^{22} + 4$	953	37	6	4
$4 \cdot 10^{23} + 4$	1001	40	7	5
$4 \cdot 10^{24} + 4$	1131	51	6	3
$4 \cdot 10^{25} + 4$	1181	70	6	5
$4 \cdot 10^{26} + 4$	1330	60	5	3
$4 \cdot 10^{27} + 4$	1304	77	10	7
$4 \cdot 10^{28} + 4$	1441	73	8	4
$4 \cdot 10^{29} + 4$	1511	94	5	3
$4 \cdot 10^{30} + 4$	1604	90	8	7
$4 \cdot 10^{31} + 4$	1754	88	3	2
$4 \cdot 10^{32} + 4$	1860	127	7	4
$4 \cdot 10^{33} + 4$	1933	174	9	8
$4 \cdot 10^{34} + 4$	2015	133	5	4
$4 \cdot 10^{35} + 4$	2090	137	8	5
$4 \cdot 10^{36} + 4$	2238	181	8	6
$4 \cdot 10^{37} + 4$	2339	214	5	4
$4 \cdot 10^{38} + 4$	2439	208	5	3
$4 \cdot 10^{39} + 4$	2483	227	9	9
$4 \cdot 10^{40} + 4$	2695	212	6	4
$4 \cdot 10^{41} + 4$	2761	260	4	3
$4 \cdot 10^{42} + 4$	2939	285	7	7
$4 \cdot 10^{43} + 4$	3112	303	5	4
$4 \cdot 10^{44} + 4$	3216	286	8	4
$4 \cdot 10^{45} + 4$	3314	323	12	12

Zur Konstruktion der Matrix X muß der Algorithmus 3.28 geringfügig modifiziert werden. Zu Beginn der Berechnung ist die Matrix X die n_2-reihige Einheitsmatrix. Bei jeder der in den Schritten (5) und (7) auf der Matrix S durchgeführten Zeilentransformationen muß nun auch die Matrix X modifiziert werden, indem auf sie die inverse Transformation angewandt wird.

Die Berechnung dieser inversen Transformation ist in allen Fällen leicht möglich, da die Zeilentransformationen immer eine spezielle Form haben. Es handelt sich, wie in Kapitel 3 erläutert wurde, jeweils um die Einbettung einer unimodularen Matrix $D \in \mathbb{Z}^{2\times 2}$ in die n_1-reihige Einheitsmatrix an geeigneten Positionen i und j. Da die Matrix D eine zweireihige unimodulare Matrix ist, ist ihr Inverses gleich ihrem Transponierten. Somit ergibt sich der folgende Algorithmus für die Modifikation der Matrix X, der während der Smith-Reduktion nach jeder Zeilenoperation durchgeführt werden muß.

Algorithmus 11.5 (Modifikation der inversen Transformationsmatrix)

Eingabe:	die Matrix $X \in \mathbb{Z}^{n_1 \times n_1}$, die Matrix $D \in \mathbb{Z}^{2\times 2}$ sowie die Positionen i und j
Ausgabe:	die modifizierte Matrix X

Vertausche die Einträge d_{12} und d_{21} der Matrix D.	(1)
Bette die Matrix D an den Positionen i und j in die n_1-reihige Einheitsmatrix ein. Es sei X' die dabei entstehende Matrix.	(2)
Setze $X \leftarrow X' \cdot X$.	(3)

11.4 Ergebnisse für die Beispieldiskriminanten

In der Tabelle auf der folgenden Seite geben wir Klassenzahl und Struktur der Klassengruppe für die 36 in den bisherigen Tabellen als Beispiele verwendeten Diskriminanten an.

Bei der Darstellung der Struktur wird die Ordnung der größten zyklischen Untergruppe nicht explizit angegeben, sondern durch einen Stern repräsentiert. Sie kann leicht berechnet werden, indem die Klassenzahl $h(\Delta)$ durch das Produkt der Ordnungen der anderen zyklische Untergruppen dividiert wird.

Man beachte, daß die Klassengruppe in allen Fällen einige sehr kleine und nur einen großen zyklischen Anteil besitzt. Die Ordnungen der kleinen Anteile sind ohne Ausnahme Zweierpotenzen. Dieses Ergebnis entspricht den heuristischen Resultaten von Cohen und Lenstra [12], nach denen der ungerade Anteil der Klassengruppe mit hoher Wahrscheinlichkeit zyklisch ist.

Die Tabelle auf der übernächsten Seite enthält die für die einzelnen Schritte des Algorithmus CLASSGROUP benötigte Rechenzeit in Sekunden. Die Angaben beziehen sich auf die Rechner SparcStation1 bzw. SparcStationSLC.

Ein Vergleich der Rechenzeiten des Algorithaus CLASSGROUP mit denen des "baby step - giant step"-Algorithmus von Shanks hat ergeben, daß der Algorithmus CLASSGROUP für $|\Delta| > 10^{25}$ deutlich schneller ist. Für diesen Vergleich wurde die im Computeralgebra-System PARI implementierte Version des Algorithmus von Shanks verwendet.

Tabelle 11.6 (Klassenzahl und Struktur der Klassengruppe)

$\lvert\Delta\rvert$	$h(\Delta)$	Struktur von $Cl(\Delta)$
$4 \cdot 10^{10} + 4$	193 584	2, 2, $*$
$4 \cdot 10^{11} + 4$	454 176	2, 2, 2, $*$
$4 \cdot 10^{12} + 4$	938 880	2, 4, $*$
$4 \cdot 10^{13} + 4$	2 968 912	2, 2, $*$
$4 \cdot 10^{14} + 4$	18 544 768	2, 2, 4, $*$
$4 \cdot 10^{15} + 4$	32 953 344	2, 2, 2, 2, 2, 4, $*$
$4 \cdot 10^{16} + 4$	188 953 856	2, 2, 2, 2, $*$
$4 \cdot 10^{17} + 4$	374 837 568	2, 2, 2, $*$
$4 \cdot 10^{18} + 4$	1 056 540 304	2, 2, $*$
$4 \cdot 10^{19} + 4$	3 298 883 812	2, $*$
$4 \cdot 10^{20} + 4$	14 849 576 832	2, 2, 2, $*$
$4 \cdot 10^{21} + 4$	43 410 764 928	2, 2, 2, 2, 2, 2, $*$
$4 \cdot 10^{22} + 4$	158 960 976 800	2, 2, 2, $*$
$4 \cdot 10^{23} + 4$	376 171 851 136	2, 2, 2, 2, $*$
$4 \cdot 10^{24} + 4$	1 154 987 161 920	2, 4, $*$
$4 \cdot 10^{25} + 4$	2 990 443 065 024	2, 2, 2, 2, $*$
$4 \cdot 10^{26} + 4$	16 503 514 325 952	2, 4, $*$
$4 \cdot 10^{27} + 4$	38 172 032 669 184	2, 2, 2, 2, 2, 2, $*$
$4 \cdot 10^{28} + 4$	159 585 440 418 304	2, 4, 4, $*$
$4 \cdot 10^{29} + 4$	436 605 442 139 680	2, 2, $*$
$4 \cdot 10^{30} + 4$	1 175 363 328 387 072	2, 2, 2, 2, 2, 8, $*$
$4 \cdot 10^{31} + 4$	4 112 798 696 458 052	2, $*$
$4 \cdot 10^{32} + 4$	17 455 078 066 482 816	2, 2, 4, $*$
$4 \cdot 10^{33} + 4$	29 696 585 481 692 160	2, 2, 2, 2, 2, 2, 4, $*$
$4 \cdot 10^{34} + 4$	189 652 590 177 168 096	2, 2, 2, $*$
$4 \cdot 10^{35} + 4$	466 984 950 044 365 952	2, 2, 2, 2, $*$
$4 \cdot 10^{36} + 4$	1 011 162 197 335 333 376	2, 2, 2, 2, 4, $*$
$4 \cdot 10^{37} + 4$	2 524 046 539 982 526 944	2, 2, 2, $*$
$4 \cdot 10^{38} + 4$	15 966 508 913 090 415 008	2, 4, $*$
$4 \cdot 10^{39} + 4$	30 094 389 081 923 893 248	2, 2, 2, 2, 2, 2, 2, 4 $*$
$4 \cdot 10^{40} + 4$	181 265 568 080 404 426 240	2, 2, 8, $*$
$4 \cdot 10^{41} + 4$	360 288 290 928 503 313 312	2, 2, $*$
$4 \cdot 10^{42} + 4$	930 179 038 714 213 854 976	2, 2, 2, 2, 2, 2, $*$
$4 \cdot 10^{43} + 4$	3 042 294 725 161 994 762 976	2, 2, 2, $*$
$4 \cdot 10^{44} + 4$	16 424 211 331 615 781 093 376	2, 4, 8, $*$
$4 \cdot 10^{45} + 4$	36 060 245 402 155 486 887 936	2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, $*$

Tabelle 11.7 (Rechenzeiten)

$\lvert\Delta\rvert$	Rechenzeiten (CPU-Sekunden) für die Schritte						
	(1) (2)	(3)	(4)	(5) (7)	(8)	(11) (12)	(13)
$4 \cdot 10^{10} + 4$	1	3	1	1	1	1	1
$4 \cdot 10^{11} + 4$	1	4	1	1	1	1	2
$4 \cdot 10^{12} + 4$	1	7	1	1	1	–	1
$4 \cdot 10^{13} + 4$	1	8	1	1	1	–	1
$4 \cdot 10^{14} + 4$	1	10	1	1	1	1	2
$4 \cdot 10^{15} + 4$	1	17	1	1	1	–	7
$4 \cdot 10^{16} + 4$	1	20	1	1	1	5	7
$4 \cdot 10^{17} + 4$	1	32	1	1	1	–	1
$4 \cdot 10^{18} + 4$	1	50	1	1	1	2	2
$4 \cdot 10^{19} + 4$	1	59	1	1	1	–	1
$4 \cdot 10^{20} + 4$	1	85	1	1	2	–	1
$4 \cdot 10^{21} + 4$	1	124	1	3	8	–	6
$4 \cdot 10^{22} + 4$	1	156	1	4	4	3	3
$4 \cdot 10^{23} + 4$	1	234	1	6	5	–	4
$4 \cdot 10^{24} + 4$	1	310	2	13	11	6	2
$4 \cdot 10^{25} + 4$	1	524	2	40	27	–	3
$4 \cdot 10^{26} + 4$	1	565	2	28	17	–	2
$4 \cdot 10^{27} + 4$	1	193	3	62	41	18	14
$4 \cdot 10^{28} + 4$	1	1013	3	51	31	9	5
$4 \cdot 10^{29} + 4$	1	1600	4	132	69	–	2
$4 \cdot 10^{30} + 4$	1	2178	5	123	68	–	5
$4 \cdot 10^{31} + 4$	2	2514	6	122	63	23	1
$4 \cdot 10^{32} + 4$	2	3734	6	398	181	51	4
$4 \cdot 10^{33} + 4$	2	7485	11	1059	370	–	8
$4 \cdot 10^{34} + 4$	3	7467	8	563	216	–	2
$4 \cdot 10^{35} + 4$	2	10527	11	675	243	94	7
$4 \cdot 10^{36} + 4$	2	15572	17	1756	543	–	6
$4 \cdot 10^{37} + 4$	2	31814	23	3496	933	89	1
$4 \cdot 10^{38} + 4$	2	26741	17	2861	835	94	2
$4 \cdot 10^{39} + 4$	2	63236	31	4428	1122	–	11
$4 \cdot 10^{40} + 4$	2	46361	22	3255	1064	160	4
$4 \cdot 10^{41} + 4$	3	95988	38	8064	2047	–	1
$4 \cdot 10^{42} + 4$	3	120604	42	9826	2615	–	4
$4 \cdot 10^{43} + 4$	3	187578	44	12887	3142	248	2
$4 \cdot 10^{44} + 4$	3	196770	50	11861	2604	–	6
$4 \cdot 10^{45} + 4$	3	344553	70	18328	3751	–	24

Kapitel 12

Verteilte Generierung der Relationen

Die Generierung der Relationen ist der bei weitem zeitaufwendigste Teil des Algorithmus CLASSGROUP. Für große Diskriminanten werden hier, wie aus Tabelle 11.7 hervorgeht, mehr als 95% der gesamten Rechenzeit verbraucht. Eine weitere Beschleunigung des Algorithmus RELATIONEN ist daher wünschenswert.

Bei einer genauen Untersuchung erkennt man, daß der Algorithmus RELATIONEN auf relativ einfache Weise parallelisierbar ist. Die einzelnen Durchläufe der Schleife (6) - (9) erfolgen nämlich relativ unabhängig voneinander, d.h. Information über bereits generierte Relationen hat nur einen sehr geringen Einfluß auf die Generierung weiterer Relationen. Um dies auszunutzen, haben wir eine Variante des Algorithmus RELATIONEN entwickelt, bei der die Generierung der Relationen auf mehreren Rechnern gleichzeitig erfolgt.

Wir bezeichnen dieses Vorgehen als **Verteilung** des Algorithmus auf mehrere Rechner bzw. als **verteilte Anwendung**. Mit dieser Bezeichnung soll die gleichzeitige Benutzung mehrerer Rechner durch einen Algorithmus von der Benutzung echter Parallelrechner abgegrenzt werden.

Wir gehen im folgenden zunächst kurz auf die grundlegende Konzeption verteilter Anwendungen ein. Anschließend beschreiben wir das von Roth und anderen [4] entwickelte System LiPS, mit dem verteilte Anwendungen auf einfache Weise realisiert werden können. Im dritten Abschnitt erläutern wir die am Algorithmus RELATIONEN für die verteilte Anwendung durchzuführenden Modifikationen. Mit dieser verteilten Anwendung wurden die Relationen zur Berechnung der Klassengruppe für eine 55-stellige Diskriminante generiert. Einige Daten von dieser Berechnung geben wir im vierten Abschnitt an.

12.1 Die Konzeption verteilter Anwendungen

Die grundlegende Motivation zur Entwicklung verteilter Anwendungen ist die Beobachtung, daß viele Workstations in großen Rechnernetzen, wie sie z.B. in Firmen und an Universitäten existieren, oft über lange Zeit hin ungenutzt bzw. nicht voll ausgelastet sind. Dies gilt insbesondere nachts und am Wochenende, wenn die Mitarbeiter in der Regel nicht an ihren Arbeitsplätzen sind. Erfahrungsgemäß sind darüberhinaus auch tagsüber immer einige Rechner nur zu einem Teil ihrer Leistungsfähigkeit ausgelastet, denn Tätigkeiten wie z.B. das Editieren von Programmen oder Textdateien erfordern fast keine CPU-Zeit.

Das Ziel verteilter Anwendungen besteht darin, die ungenutzte CPU-Zeit aller Rechner in einem Netz für rechenzeitintensive Algorithmen nutzbar zu machen. Es wird also nicht eigens ein Rechnernetz für verteilte Anwendungen bereitgestellt, sondern es werden die Leerlaufzeiten in einem vorhandenen Netz benutzt, dessen einzelne Rechner primär anderen Aufgaben dienen. Die Nutzung der beteiligten Rechner für ihre eigentlichen Aufgaben darf daher durch verteilte Anwendungen in keiner Weise beeinträchtigt werden.

Diese Idee ist nicht neu. Bereits im Jahre 1987 wurde sie von R.D. Silverman [42] für eine verteilte Implementierung des Faktorisierungsalgorithmus "Multiple Polynomial Quadratic Sieve" verwendet. Ihm gelang so erstmals die Faktorisierung einer 87-stelligen Zahl. Weitere Beispiele sind in [4] angegeben.

Offensichtlich wird durch die Verteilung eines Algorithmus auf mehrere Rechner die insgesamt verbrauchte CPU-Zeit nicht herabgesetzt, wohl aber die "Warte"zeit bis zum Abschluß einer Berechnung. Die Zeitverkürzung hängt ab von der Anzahl der beteiligten Rechner sowie der auf den einzelnen Rechnern für die verteilte Anwendung zur Verfügung stehenden Rechenzeit.

Voraussetzung für die Verteilbarkeit eines Algorithmus ist es, daß die einzelnen Programme auf den verschiedenen Rechnern relativ unabhängig voneinander ablaufen können, so daß nur eine sehr geringe Kommunikation zwischen den Rechnern notwendig ist. Andernfalls würde das benutzte Netzwerk zu stark belastet. Diese Voraussetzung ist, wie sich im nächsten Abschnitt zeigen wird, für die Generierung der Relationen im Algorithmus CLASSGROUP erfüllt.

Üblicherweise basieren verteilte Anwendungen auf dem sogenannten Server-Klienten-Konzept. Ein ausgezeichneter Rechner, der als Server bezeichnet wird, ist dabei für die Steuerung des Gesamtablaufs zuständig und fügt die auf den übrigen Rechnern ermittelten Ergebnisse zusammen. Die übrigen Rechner, die sogenannten Klienten, arbeiten unabhängig voneinander - gemäß der Steuerung durch den Server - und machen die von ihnen berechneten Ergebnisse dem Server zugänglich.

Auf allen Klienten läuft in der Regel dasselbe Programm. Dabei muß sichergestellt werden, daß nicht mehrere Klienten übereinstimmende Ergebnisse liefern, die im Rahmen des Gesamtalgorithmus nicht mehrmals nutzbar sind. Dies geschieht mit Hilfe unterschiedlicher Eingaben des Servers an die einzelnen Klienten-Prozesse oder durch die Abhängigkeit dieser Programme von zufälligen Berechnungen.

Zur Realisierung einer verteilten Anwendung bedarf es also der Aufteilung des zugrundeliegenden Ausgangs-Algorithmus in einen Server-Algorithmus und einen Klienten-Algorithmus sowie der Festlegung der zwischen dem Server und den Klienten notwendigen Kommunikation.

In unserer Anwendung haben die Klienten die Aufgabe, die einzelnen Relationen zu generieren. Der Server fügt diese Relationen zusammen und sorgt durch geeignete Steuerung der Klienten-Programme dafür, daß die Gesamtheit der Relationen nach Beendigung der Berechnung die gewünschten Eigenschaften aufweist. Auf Einzelheiten dieser Aufgabenteilung und die dadurch bedingte Kommunikation zwischen dem Server und den Klienten gehen wir im übernächsten Abschnitt ein.

Zur Realisierung einer verteilten Anwendung sind über die eigentliche Aufteilung des zugrundeliegenden Algorithmus hinaus noch eine Reihe anderer Probleme zu lösen. So muß z.B. die Kommunikation zwischen zwei nicht auf demselben Rechner ablaufenden Programmen ermöglicht werden. Ferner muß sichergestellt werden, daß die verteilte Anwendung den normalen Betrieb auf den einzelnen beteiligten Rechnern nicht beeinträchtigt.

Diese Probleme sind von dem zu verteilenden Algorithmus völlig unabhängig. Zu ihrer generellen Lösung wurde von Roth [4] das System LiPS (**Li**brary for **P**arallel **S**ystems) entwickelt. Es handelt sich dabei um eine Bibliothek von C-Funktionen zur Prozeß-Kommunikation und Prozeß-Steuerung für verteilte Anwendungen.

Wir verwenden zur Realisierung unserer verteilten Anwendung Funktionen aus LiPS. Daher beschäftigen wir uns im folgenden näher mit diesem System. Wir beschränken uns dabei jedoch auf die Darstellung der grundlegenden Eigenschaften von LiPS, soweit sie für unsere Anwendung von Bedeutung sind.

12.2 Grundlegende Eigenschaften von LiPS

LiPS erlaubt die Verteilung eines Algorithmus über ein Netz von Workstations, die mit dem UNIX-Betriebssystem BSD 4.2 bzw. mit einem dazu weitgehend kompatiblen Betriebssystem (z.B. SunOS 4.1) betrieben werden. Ferner wird die Vernetzung der beteiligten Workstations über Ethernet vorausgesetzt.

Darüberhinaus werden an die Konfiguration der einzelnen beteiligten Rechner keine besonderen Bedingungen gestellt. Damit wird die Tatsache berücksichtigt, daß die einzelnen zu benutzenden Rechner eventuell zu unterschiedlichen Arbeitsgruppen gehören. Sie unterstehen daher teilweise der Verantwortung verschiedener Systemverwalter und sind - den Bedürfnissen der jeweiligen Arbeitsgruppe entsprechend - unterschiedlich konfiguriert.

So wird insbesondere nicht vorausgesetzt, daß von allen beteiligten Rechnern aus direkter Zugriff auf eine globale Festplatte besteht. Die Erfüllung einer solchen Bedingung würde im übrigen - neben der einschränkenden Voraussetzung an die Rechnerkonfiguration - eine sehr starke Belastung des Netzes verursachen, die ebenfalls vermieden werden soll.

LiPS ist ferner so gestaltet, daß weder die Installation noch der Ablauf verteilter Anwendungen Privilegien erfordert, die über die normalen Rechte eines Benutzers auf einer Workstation hinausgehen. Es ist also nicht erforderlich, daß sich die Systemverwalter der zu benutzenden Rechner mit der Verwaltung von LiPS beschäftigen. Um einen Rechner für verteilte Anwendungen nutzbar zu machen, ist lediglich eine gewöhnliche Benutzerkennung mit minimalen Rechten zu vergeben. Dies ist - aus Sicherheitsgründen aus der Sicht der einzelnen Systemverwalter - unabdingbare Voraussetzung dafür, auf Rechnern fremder Arbeitsgruppen die Erlaubnis zur Durchführung verteilter Anwendungen zu erhalten.

Eine weitere sehr wichtige Anforderung an eine verteilte Anwendung besteht, wie bereits erwähnt, darin, daß die Nutzung der beteiligten Rechner für ihre eigentlichen Aufgaben in keiner Weise behindert werden darf. Um dies zu gewährleisten, laufen verteilte Anwendungen, die LiPS benutzen, automatisch mit der niedrigsten möglichen Priorität ab.

LiPS ermöglicht es darüberhinaus, die verteilte Anwendung unter bestimmten Bedingungen, die sich für jeden beteiligten Rechner unterscheiden können, völlig anzuhalten. Dazu werden vom jeweiligen Systemverwalter Zeitspannen definiert, in denen die verteilte Anwendung

- ohne Berücksichtigung der Anzahl angemeldeter Benutzer rechnen darf,
- nur rechnen darf, wenn eine bestimmte Anzahl von Benutzern nicht überschritten wird,
- nur rechnen darf, wenn alle angemeldeten Benutzer eine gewisse Zeit lang keine Eingabe gemacht haben,
- überhaupt nicht rechnen darf.

Während der Laufzeit einer verteilten Anwendung wird nun auf jedem Klienten-Rechner in regelmäßigen Abständen durch einen Kontrollprozeß überprüft, ob noch Rechenberechtigung besteht. Ist dies nicht der Fall, so wird das auf diesem Rechner laufende Programm der verteilten Anwendung angehalten. Sobald der Kontrollprozeß zu einem späteren Zeitpunkt feststellt, daß die Rechenberechtigung wieder gegeben ist, wird das entsprechende Programm wieder aktiviert.

Die auf diese Weise erzielte Flexibilität gewährleistet, daß die Belastung der einzelnen Klienten-Rechner durch die verteilte Anwendung vollständig den Vorstellungen der jeweiligen Systemverwalter angepaßt werden kann. Dies ist ebenfalls eine wichtige Voraussetzung dafür, auf möglichst vielen Rechnern eines Netzes die Erlaubnis zur Durchführung verteilter Anwendungen zu erhalten.

Diese Flexibilität hat jedoch ebenfalls Auswirkungen auf die Aufteilung des zugrundeliegenden Algorithmus. Man kann nämlich nicht davon ausgehen, daß jeder Klienten-Rechner mit demselben Kontingent an Rechenzeit an der verteilten Berechnung beteiligt ist und insofern in gleichem Umfang zum Ergebnis der Berechnung beiträgt. Es ist im Extremfall denkbar, daß während der gesamten Laufzeit einer verteilten Anwendung auf einem der vorgesehenen Klienten-Rechner zu keiner Zeit Rechenberechtigung besteht. Man darf also im Zuge der Verteilung nicht einzelnen Klienten-Rechnern fest definierte Aufgaben zuordnen, von deren Lösung die Beendigung des Gesamtalgorithmus abhängt.

Wir beschreiben nun kurz die von LiPS für die Kommunikation zwischen dem Server und den Klienten zur Verfügung gestellten Mechanismen. Dabei ist zwischen zwei grundsätzlich verschiedenen Arten der Kommunikation zu unterscheiden.

Zum einen ist es möglich, Dateien zwischen verschiedenen Rechnern zu kopieren. Solche Dateien müssen zunächst von einem der an der verteilten Anwendung beteiligten Programme explizit angelegt werden, können dann mit Hilfe von LiPS-Funktionen auf einen oder mehrere andere Rechner kopiert werden und müssen schließlich von den dort laufenden Programmen wieder explizit eingelesen werden. Diese Form der Kommunikation ist somit insbesondere für die seltene Übermittlung großer Datenmengen geeignet. Sie wird in unserer Anwendung zur Übertragung der Formenbasis auf alle Klienten-Rechner genutzt.

Die zweite Möglichkeit der Kommunikation besteht im direkten Austausch von Nachrichten zwischen den Programmen, die auf den einzelnen Rechnern ablaufen. Solche Kommunikationskanäle zwischen verschiedenen Prozessen sowohl auf einem Rechner als auch auf verschiedenen Rechnern werden im Betriebssystem UNIX als Socket-Streams bezeichnet. Die in LiPS hierzu bereitgestellten Funktionen umfassen den Auf- und Abbau der gewünschten Verbindung sowie das eigentliche Übertragen der Nachricht. Ferner werden eingehende Nachrichten automatisch so lange gepuffert, bis sie von dem entsprechenden Anwendungsprogramm gelesen und abgearbeitet werden. Diese Art der Kommunikation eignet sich für die häufigere Übermittlung sehr kleiner Datenmengen. In unserer Anwendung werden die einzelnen auf den Klienten-Rechnern generierten Relationen auf diese Weise zum Server übertragen. Diese Art der Kommunikation wird außerdem für die Übermittlung von Befehlen zur Steuerung der Klienten durch den Server genutzt.

Zu Beginn einer verteilten Anwendung startet der Benutzer das Server-Programm, das sich zunächst nicht von einem gewöhnlichen C-Programm unterscheidet. Durch einen bestimmten Funktionsaufruf innerhalb des Server-Programms erfolgt dann der eigentliche Start der Verteilung, indem sich ein Kontrollprozeß vom Server-Programm abspaltet. Die Aufgabe dieses Kontrollprozesses besteht zunächst darin, Verbindungen zu den einzelnen als Klienten vorgesehenen Rechnern aufzubauen sowie das zum Ablauf auf den Klienten vorgesehene Programm auf diese Rechner zu kopieren und dann dort zu starten. Die dazu benötigten Informationen, z.B. eine Liste der als Klienten vorgesehenen Rechner und der Name des dort zu startenden Programms, werden aus einer Textdatei eingelesen, die der Benutzer vor Beginn der verteilten Anwendung erstellen muß.

Nach dem Start der Klienten-Programme spaltet sich auch auf jedem Klienten-Rechner ein Kontrollprozeß ab. Auf jedem der an der verteilten Anwendung beteiligten Rechner existieren nun zwei Prozesse, nämlich das eigentliche Anwendungsprogramm und der Kontrollprozeß. Zwischen diesen beiden Prozessen wird eine permanente Socket-Verbindung aufrecht erhalten.

Die Hauptaufgabe der Kontrollprozesse besteht in der Abwicklung der Kommunikation zwischen den Anwendungsprogrammen auf den verschiedenen Rechnern. Wir erläutern dies im folgenden am Beispiel des Nachrichtenaustauschs über eine Socket-Verbindung.

Eines der Anwendungsprogramme gibt dazu z.B. die Anforderung zum Versenden einer Nachricht an einen bestimmten Zielrechner über die permanente Socket-Verbindung an seinen Kontrollprozeß weiter. Dieser baut nun eine temporäre Socket-

Verbindung zum Kontrollprozeß auf dem Zielrechner auf und überträgt die Nachricht. Der Kontrollprozeß auf dem Zielrechner puffert die Nachricht und gibt sie - wiederum über die permanente Socket-Verbindung - an sein Anwendungsprogramm weiter, wenn dieses eine entsprechende Anfrage stellt.

Man beachte, daß die Socket-Verbindungen zwischen den Rechnern nur temporär für die Dauer einer Kommunikation aufgebaut werden. Permanente Socket-Verbindungen zwischen dem Server und allen Klienten würden die Netzbelastung zu stark erhöhen.

Man beachte ferner, daß die Kontrollprozesse kaum Rechenzeit benötigen, da sie im wesentlichen nur dann aktiv sind, wenn Kommunikationsanforderungen abzuarbeiten sind.

Die Kontrollprozesse auf den Klienten-Rechnern haben neben der Abwicklung der Kommunikation die Aufgabe, die oben beschriebene Rechenberechtigung zu überwachen und die Klienten-Programme anzuhalten bzw. zu reaktivieren, wenn sich die Rechenberechtigung ändert. Während der inaktiven Phasen eines Klienten-Programms puffert der entsprechende Kontrollprozeß alle vom Server eingehenden Nachrichten, um sie später, wenn das Klienten-Programm wieder aktiv ist, an dieses weiterzugeben. Man beachte in diesem Zusammenhang, daß die Kontrollprozesse selbst nicht inaktiv werden. Wegen des sehr geringen Rechenzeitbedarfs der Kontrollprozesse wird dadurch jedoch das normale Arbeiten an den beteiligten Rechnern nicht behindert.

Die Beendigung der verteilten Anwendung erfolgt, indem das Server-Programm eine entsprechende Nachricht an die Kontrollprozesse auf den Klienten-Rechnern verschickt. Diese beenden daraufhin zunächst die Klienten-Programme und dann sich selbst. Danach wird auch der Kontrollprozeß auf dem Server beendet, womit der Anfangszustand wieder hergestellt ist, d.h. es läuft nur noch das ursprünglich vom Benutzer gestartete Server-Programm.

12.3 Die Verteilung des Algorithmus RELATIONEN

Wir beschreiben in diesem Abschnitt Aufbau und Arbeitsweise der Algorithmen zur verteilten Generierung der Relationen. Dazu ist das im Algorithmus RELATIONEN angegebene Verfahren in geeigneter Weise in einen Server- und einen Klienten-Algorithmus aufzuteilen. Die eigentliche Generierung von Relationen soll dabei auf den Klienten-Rechnern erfolgen; der Server soll die Relationen zusammenfügen sowie den Gesamtablauf steuern.

Die Generierung der Relationen vollzieht sich im Algorithmus RELATIONEN in der Schleife (6) - (9). Diese Schleife bildet also den zentralen Teil des Klienten-Algorithmus.

Man beachte, daß nach Einführung der Large-Prime-Variante in dieser Schleife sowohl vollständige als auch unvollständige Relationen entstehen. Letztere sind dadurch gekennzeichnet, daß sie nur in Kombination mit einer geeigneten weiteren unvollständigen Relation eine vollständige Relation ergeben und damit für den Gesamtalgorithmus nutzbar werden (vgl. hierzu die Ausführungen in Abschnitt 8.5).

Nun ist in der Regel nicht zu erwarten, daß zwei unvollständige Relationen, die sich kombinieren lassen, auf demselben Klienten-Rechner generiert werden. Man muß vielmehr damit rechnen, daß solche kombinierbaren Relationen auf verschiedenen Klienten entstehen. Daher ist es erforderlich, die Sammlung der unvollständigen Relationen, die Überprüfung ihrer Kombinierbarkeit sowie die eigentliche Kombination zentral auf dem Server durchzuführen. Die Schritte (11) bis (15) des Algorithmus RELATIONEN sind somit Bestandteil des Server-Algorithmus.

Nach diesen Überlegungen steht die Aufgabenteilung zwischen dem Server und den Klienten und damit auch die für die verteilte Anwendung notwendige Kommunikationsstruktur im wesentlichen fest. Die Klienten-Rechner generieren vollständige und unvollständige Relationen und übermitteln diese an den Server. Dieser sammelt die von den einzelnen Klienten eingehenden Relationen und führt die Kombination unvollständiger Relationen zu vollständigen Relationen durch.

Die Generierung der Relationen muß auch in der verteilten Anwendung so gesteuert werden, daß die entstehende Relationenmatrix mit hoher Wahrscheinlichkeit nicht singulär ist. In der sequentiellen Version des Algorithmus dient dazu die Menge S, die jeweils die Nummer derjenigen Faktorbasis-Elemente enthält, die in den bisher vorhandenen Relationen noch nicht in geeigneter Weise mit einem von Null verschiedenen Eintrag repräsentiert sind (vgl. hierzu die Ausführungen in Abschnitt 8.2).

Diese Menge S muß in der verteilten Anwendung allen Klienten-Rechnern bekannt sein und ständig aktualisiert werden. Dazu stellt der Server jedesmal, wenn er eine vollständige Relation von einem Klienten erhält oder wenn er zwei unvollständige Relationen kombinieren konnte, den aus S zu entfernenden Index fest und teilt ihn allen Klienten-Rechnern mit. Neben der Übertragung der einzelnen Relationen von den Klienten auf den Server ist also während des gesamten Algorithmus auch die Übertragung von Nachrichten vom Server zu allen Klienten erforderlich.

Eine letzte Überlegung betrifft die zur Initialisierung der Algorithmen notwendigen Schritte. Zu Beginn werden zunächst die absolute Diskriminante D sowie die Faktorbasis F und deren Größe n vom Server aus allen Klienten-Rechnern übermittelt. Alle weiteren für die Generierung der Relationen notwendigen Konstanten werden ebenfalls zentral vom Server festgelegt und allen Klienten mitgeteilt. Lediglich die Vorberechnung der häufig benötigten Potenzen von Formenklassen erfolgt auf jedem Klienten, da dies insgesamt weniger aufwendig ist als die zentrale Berechnung dieser Daten auf dem Server und die anschließende Übertragung zu allen Klienten.

Insgesamt ergeben sich die im folgenden angegebenen Algorithmen für den Server und die Klienten.

Algorithmus 12.1 (Rahmenalgorithmus RELATIONEN-SERVER)

Eingabe: Δ, F, F_p, n, k und M wie in Kapitel 8 angegeben
Ausgabe: eine Matrix $A \in \mathbb{Z}^{n\times m}$ mit $m \geq n+5$, deren Spalten Relationen über F sind

/* Initialisierung der Konstanten */				
Setze $k_0 \leftarrow \max\Big(\{1 \leq i \leq k_1 \ : \ p_i < \log\lvert\Delta\rvert\} \cup \{4\}\Big)$, $t \leftarrow \max\{4, \lfloor \frac{k_0}{2} \rfloor\}$ und $B \leftarrow 30$.				(1)
Setze $k_1 \leftarrow \lfloor \frac{n}{4} \rfloor$. Setze $k_2 \leftarrow \max\{\lfloor \frac{n}{2} \rfloor, n'\}$, wobei $n' = \min\{1 \leq i \leq k_g \ : p_i^2 > p_{k_g}\}$ ist.				(2)
Setze $S \leftarrow \{1, \ldots, n\}$, $k \leftarrow 1$ und $L \leftarrow p_{k_2}^2$.				(3)
/* Initialisierung der verteilten Anwendung */				
Veranlasse den Start aller Klienten-Programme sowie das Kopieren der Datei, die die Formenbasis F enthält, an alle Klienten.				(4)
Sende eine Nachricht mit $D, n, k, k_0, t, B, k_1, k_2, L$ an alle Klienten.				(5)
/* Verwaltung der eingehenden Relationen */				
WHILE $k \leq n+5$ oder $S \neq \emptyset$				(6)
	Warte auf eine Nachricht (r, w, b) von einem der Klienten.			(7)
	IF $r > 1$			(8)
	THEN	IF in der Liste der unvollständigen Relationen existiert ein Eintrag (r', w', b') mit $r' = r$		(9)
		THEN	Berechne aus (r, w, b) und (r', w', b') eine vollständige Relation $v \in \mathbb{Z}^n$. (Teilagorithmus LARGEREL)	(10)
			Setze $w \leftarrow v$ und $r = 1$.	(11)
		ELSE	Speichere (r, w, b) in der Liste der unvollständigen Faktorisierungen.	(12)
	IF $r = 1$ und w ist nicht der Nullvektor			(13)
	THEN	IF $S' = \{\nu \in S \ : w_\nu \neq 0\} \neq \emptyset$		(14)
		THEN	Setze $j = \max S'$ und entferne j aus S.	(15)
			Sende eine Nachricht j an alle Klienten.	(16)
		Setze (Spalte) $A_k \leftarrow w$ und $k \leftarrow k+1$.		(17)
/* Beendigung der verteilten Anwendung */				
Veranlasse die Beendigung aller Klienten-Programme.				(18)

Algorithmus 12.2 (Rahmenalgorithmus RELATIONEN-KLIENT)

Eingabe: Kommunikation mit dem Server-Algorithmus
Ausgabe: Kommunikation mit dem Server-Algorithmus

Lies die vom Server übermittelte Datei mit der Formenbasis F ein und empfange die Nachricht mit den Parametern $D, n, k, k_0, t, B, k_1, k_2, L$ vom Server.			(1)
Berechne die reduzierten Formen h_{ij} in den Klassen $[f_i^j]$ für $1 \leq i \leq k_0$, $1 \leq j \leq B$. (Teilalgorithmus INITPOTENZ)			(2)
Setze $S = \{1, \dots n\}$.			(3)
LOOP			(4)
	Wähle $s \in S$ zufällig.		(5)
	Wähle einen Exponentenvektor $x = (x_1, \dots, x_n) \in \mathbb{Z}^n$ mit $x_s \neq 0$ und berechne die reduzierte Form $g = (a, b)$ in der Klasse von $\prod_{i=1}^{n} f_i^{x_i}$. (Teilalgorithmus POTENZPROD)		(6)
	Stelle fest, ob a vollständig über F_p zerfällt und, falls ja, berechne den Zerlegungsvektor $y = (y_1, \dots, y_n) \in \mathbb{Z}^n$ mit $a = \prod_{i=1}^{n} p_i^{y_i}$. Gib ferner den unzerlegbaren Anteil r von a an. (Teilalgorithmus FAKTOR)		(7)
	IF $r \leq C_4$.		(8)
	THEN	Berechne den (vollständigen oder unvollständigen) Relationenvektor $w = (w_1, \dots w_n) \in \mathbb{Z}^n$. (Teilalgorithmus RELVEKTOR)	(9)
		Sende (w, r, b) als Nachricht an den Server.	(10)
	IF inzwischen ist die Nachricht zum Beenden der Klienten-Programme vom Server angekommen THEN EXIT.		(11)
	IF inzwischen sind eine oder mehrere andere Nachrichten vom Server angekommen		(12)
	THEN	Für jede solche Nachricht j setze $S = S \setminus \{j\}$.	(13)

Man vergleiche hierzu auch die einzelnen Schritte des Algorithmus REALTIONEN. Alle Teilalgorithmen können unverändert aus der sequentiellen Version übernommen werden.

Wir weisen noch darauf hin, daß es, insbesondere wenn die Menge S bereits sehr klein ist, durchaus vorkommen kann, daß zwei Klienten in Schritt (5) von Algorithmus 12.2 zur selben Zeit denselben Index $i \in S$ wählen. Dies hat sich jedoch in der Praxis nicht als nachteilig für den Gesamtablauf erwiesen. Es führt lediglich dazu, daß insgesamt geringfügig mehr als $n + 5$ Relationen generiert werden, bevor die Menge S leer ist und damit der Server-Algorithmus das Abbruchkriterium erreicht.

12.4 Effizienz der verteilten Anwendung

Als Beispiel für die Effizienz der verteilten Generierung der Relationen geben wir im folgenden einige Daten von der Berechnung der Klassengruppe für eine 55-stellige Diskriminante an.

Tabelle 12.3 (Beispiel für die verteilte Anwendung)

absolute Diskriminante $\|\Delta\|$	$4 \cdot 10^{54} + 4$
Klassenzahl $h(\Delta)$	1 056 175 002 108 254 379 317 829 632
Sruktur der Klassengruppe $Cl(\Delta)$	2, 2, 2, 2, 2, *

Die Formenbasis F bestand für diese Diskriminante aus 4585 Elementen, es mußten also mindestens 4590 Relationen generiert werden. Dazu wurde mit den Algorithmen RELATION-SERVER und RELATION-KLIENT eine Zeit von etwa 282 Stunden benötigt. An der Berechnung waren 14 Klienten-Rechner (SparcStation1 und SparcStationSLC) beteiligt.

In der folgenden Tabelle geben wir einige interessante Daten von dieser verteilten Generierung der Relationenmatrix an. Zu jedem Klienten-Rechner wird zunächst in der zweiten Spalte die verbrauchte CPU-Zeit (in Stunden) angegeben. Hier erkennt man, daß auf den einzelnen Rechnern innerhalb des Zeitraums, in dem die Berechnung stattfand, unterschiedlich viel ungenutzte CPU-Zeit zur Verfügung stand. In der dritten Spalte geben wir die Anzahl der auf den einzelnen Klienten-Rechnern getesteten Exponentenvektoren an. Die vierte Spalte schließlich enthält die Anzahl der Erfolge, d.h. die Anzahl der vollständigen oder unvollständigen Relationen, die dem Server von den einzelnen Klienten übermittelt wurden.

Tabelle 12.4 (Verteilte Generierung der Relationen)

Name des Klienten-Rechners	Rechenzeit (in Stunden)	Anzahl der getesteten Exp.vektoren	Anzahl der Erfolge
crypt1	149.4	$3.18 \cdot 10^6$	665
crypt2	213.8	$4.58 \cdot 10^6$	953
cauchy	281.9	$6.91 \cdot 10^6$	1412
fermat	279.8	$6.86 \cdot 10^6$	1390
fourier	85.3	$2.17 \cdot 10^6$	423
gauss	281.1	$6.89 \cdot 10^6$	1395
hilbert	281.2	$6.86 \cdot 10^6$	1392
laplace	276.2	$6.78 \cdot 10^6$	1363
riemann	281.5	$6.86 \cdot 10^6$	1431
batman	234.3	$4.83 \cdot 10^6$	983
robin	125.3	$2.46 \cdot 10^6$	529
p1sun	212.8	$5.37 \cdot 10^6$	1037
p2sun	125.2	$3.07 \cdot 10^6$	635
cadsun	104.5	$2.69 \cdot 10^6$	523

Der Algorithmus RELATIONEN-SERVER wurde ebenfalls auf dem Rechner *crypt1* durchgeführt. Er benötigte eine Rechenzeit von 5.7 Stunden.

Die Summe der auf den einzelnen Klienten-Rechnern für die Generierung der Relationen aufgewandten Rechenzeit betrug 2932.3 Stunden, also etwa 18 Wochen. Ohne die verteilte Anwendung wäre die Berechnung dieser Klassengruppe somit nicht in vertretbarer Zeit durchführbar gewesen.

In der folgenden Tabelle geben wir die Laufzeit der übrigen Schritte des Algorithmus CLASSGROUP für diese Diskriminante an.

Tabelle 12.5 (Laufzeit der übrigen Schritte)

	Rechenzeit
Schritte (1), (2)	7 Sekunden
Schritt (3)	siehe oben
Schritt (4)	176 Sekunden
Schritt (5)	23.4 Stunden
Schritt (8)	3.9 Stunden
Schritt (13)	3 Sekunden

Die Generierung zusätzlicher Relationen in den Schritten (7) oder (11) des Algorithmus CLASSGROUP war nicht erforderlich.

Die in Schritt (3) generierte usprüngliche Relationenmatrix A hatte eine Größe von 4885 Zeilen und 4894 Spalten. Die in Schritt (4) konstruierte verkleinerte Relationenmatrix B enthielt nur noch 473 Zeilen und 482 Spalten. Bei der Smith-Reduktion in Schritt (13) mußte nur noch eine Matrix mit 6 Zeilen und Spalten bearbeitet werden.

Anhang: Klassengruppentabelle

In der folgenden Tabelle werden Klassenzahl und Struktur der Klassengruppe für eintausend 30-stellige Diskriminanten angegeben. Es handelt sich um die Diskriminanten

$$\Delta = -(4 \cdot 10^{29} + d)$$

mit

$$d \in [\![0, 1999]\!], \qquad d \equiv 0, 3 \bmod 4.$$

Alle Berechnungen wurden mit dem Algorithmus CLASSGROUP durchgeführt. Die Rechenzeit betrug im Durchschnitt 40 Minuten pro Diskriminante auf einer SparcStationSLC.

Die Struktur der Klassengruppe wird in derselben Weise dargestellt wie in Tabelle 11.6, d.h. anstelle der Ordnung des größten zyklischen Anteils wird in der Tabelle ein Stern angegeben. Man beachte, daß in fast allen Fällen nur diese Ordnung ungerade Primfaktoren enthält. Die übrigen Ordnungen sind in der Regel Zweierpotenzen. Dieses Ergebnis entspricht wiederum den heuristischen Resultaten von Cohen und Lenstra [12].

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
0	100 000 000 000 000	2, $*$
3	35 268 603 546 136	2, $*$
4	157 556 903 813 360	2, 2, 2, $*$
7	374 024 358 963 900	2, $*$
8	129 736 229 109 312	2, 2, 2, 2, 2, $*$
11	66 574 441 505 856	2, 2, 2, $*$
12	70 616 922 829 116	2, $*$
15	176 104 021 428 776	2, $*$
16	147 050 346 094 080	2, 4, 4, $*$
19	189 477 776 277 672	2, 2, $*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
20	105 157 207 850 112	2, 2, 2, 2, ∗
23	187 560 472 318 032	2, 2, 2, ∗
24	80 159 581 263 072	2, 2, 2, 2, ∗
27	32 160 826 742 816	2, 2, 2, ∗
28	110 675 182 948 544	2, 2, ∗
31	360 486 593 080 160	2, 2, ∗
32	86 876 251 758 144	2, 2, 2, 2, ∗
35	61 482 925 131 840	2, 2, 2, 2, 2, ∗
36	123 493 748 990 212	2, ∗
39	199 861 944 151 422	2, 2, 2, 2, 2, ∗
40	157 328 081 408 320	2, 2, 2, 2, 2, ∗
43	83 491 848 689 848	2, 2, ∗
44	168 852 973 449 312	2, 2, ∗
47	249 309 765 657 904	2, ∗
48	52 569 286 192 128	2, 2, 2, 2, 2, ∗
51	85 314 790 632 040	2, 2, ∗
52	114 425 637 241 388	2, ∗
55	333 181 926 049 312	2, 2, 2, 2, ∗
56	138 365 942 268 608	2, 2, 2, 2, ∗
59	64 704 247 428 312	2, 2, ∗
60	69 793 898 224 800	2, 2, 2, ∗
63	109 100 365 599 856	2, 2, 2, ∗
64	280 199 966 140 960	2, 2, ∗
67	54 639 515 017 444	2, ∗
68	78 598 241 010 912	2, 2, 2, ∗
71	294 471 793 365 992	2, ∗
72	55 783 287 573 256	2, ∗
75	80 216 418 217 536	2, ∗
76	185 138 342 053 200	2, ∗
79	201 105 297 291 376	2, 2, ∗
80	82 093 666 214 400	2, 2, 2, 4, ∗
83	54 391 454 982 208	2, 4, ∗
84	133 449 078 982 832	2, 2, 2, ∗
87	95 251 235 650 840	2, 2, ∗
88	131 884 941 411 540	2, ∗
91	194 803 539 386 456	2, 2, ∗
92	78 939 515 045 032	2, ∗
95	218 395 860 002 296	2, 2, ∗
96	130 321 384 336 928	2, 2, 2, ∗
99	99 830 925 913 068	∗
100	152 688 130 676 736	2, 2, 2, 2, 2, 2, 2, ∗
103	239 363 108 952 512	8, ∗
104	112 125 678 265 824	2, 2, ∗
107	55 294 865 612 124	2, ∗
108	57 476 805 683 568	2, 2, 2, ∗

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
111	154 857 196 312 352	2, 2, 2, $*$
112	166 669 815 392 928	6, $*$
115	82 148 080 516 688	2, 2, 2, $*$
116	102 990 136 308 192	2, 2, $*$
119	306 669 343 646 400	2, 2, 2, 2, $*$
120	73 061 709 915 520	2, 2, $*$
123	34 718 846 940 000	2, 2, $*$
124	200 764 037 179 712	2, 2, 2, $*$
127	277 712 403 636 624	2, 2, 2, $*$
128	79 663 410 611 840	2, 2, 2, 2, 2, $*$
131	90 753 864 771 648	2, 2, 2, $*$
132	55 504 698 513 944	2, $*$
135	155 437 574 759 456	2, 2, 2, $*$
136	136 869 903 297 476	2, $*$
139	110 855 317 534 556	2, $*$
140	150 370 329 775 872	2, 2, 2, 4, $*$
143	189 611 066 268 144	2, $*$
144	76 898 210 569 432	2, $*$
147	32 242 456 600 200	2, $*$
148	157 784 633 231 646	$*$
151	405 750 671 813 480	2, $*$
152	99 591 036 346 944	2, 2, 2, 2, $*$
155	82 517 946 038 416	2, 2, 2, $*$
156	83 161 292 068 080	2, 6, $*$
159	207 638 092 314 000	2, 2, $*$
160	131 602 550 077 120	2, 2, 2, 2, $*$
163	90 165 576 660 360	$*$
164	98 656 683 046 720	2, 2, 2, 2, 2, $*$
167	145 159 835 837 818	$*$
168	103 569 106 989 696	2, 2, 2, $*$
171	44 146 147 727 974	$*$
172	126 578 687 626 332	6, $*$
175	432 763 052 438 292	2, $*$
176	135 268 756 118 976	2, 2, 2, 2, $*$
179	72 712 240 155 876	$*$
180	90 678 438 282 768	2, 2, $*$
183	121 581 465 304 768	4, $*$
184	258 341 616 075 984	2, 2, 2, $*$
187	111 959 049 508 872	2, $*$
188	92 320 984 773 504	2, 2, 12, $*$
191	312 786 879 187 424	2, 2, 2, 2, $*$
192	65 810 370 385 920	2, 2, 2, 4, 4, $*$
195	50 608 972 610 960	2, 2, $*$
196	181 819 777 610 736	2, 2, $*$
199	268 663 792 745 600	2, 2, $*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
200	93 975 157 364 592	2, 2, 2, $*$
203	64 073 968 134 378	$*$
204	90 835 570 705 152	2, 4, $*$
207	138 442 366 258 704	2, 2, 2, $*$
208	167 206 470 310 224	2, 2, $*$
211	119 748 652 543 680	2, 2, $*$
212	86 886 067 234 304	2, 2, $*$
215	186 177 485 225 868	2, $*$
216	106 829 610 711 552	2, 2, 2, $*$
219	45 465 331 019 040	2, 2, 2, 2, $*$
220	122 585 368 390 560	2, 2, 2, 2, $*$
223	256 262 721 880 964	2, $*$
224	115 455 029 567 136	2, 2, 2, $*$
227	65 846 576 166 480	2, 2, $*$
228	61 884 813 307 844	2, $*$
231	274 138 731 190 400	2, 2, 2, $*$
232	140 994 709 187 776	2, 2, 2, $*$
235	100 651 800 411 392	2, $*$
236	137 119 446 405 312	2, 2, 2, $*$
239	272 115 232 335 680	2, 2, $*$
240	79 638 651 557 600	2, 2, 2, 2, $*$
243	56 953 778 689 552	2, 2, $*$
244	161 945 203 602 192	2, 2, $*$
247	247 116 939 015 040	2, 4, $*$
248	64 787 325 362 064	2, 2, $*$
251	72 408 219 456 768	2, 2, 2, 2, $*$
252	70 905 319 696 476	2, $*$
255	131 150 975 725 568	2, 2, 2, 2, $*$
256	170 945 545 898 880	2, 4, $*$
259	167 290 612 393 132	$*$
260	174 967 928 933 504	2, 2, 2, 2, 2, $*$
263	118 383 487 361 920	2, 2, 2, $*$
264	94 529 533 634 240	2, 2, 2, $*$
267	59 615 200 435 620	2, $*$
268	108 209 533 076 196	$*$
271	452 604 432 534 864	2, 2, 2, $*$
272	71 786 682 763 520	2, 2, 2, 2, 4, $*$
275	60 237 269 567 520	2, 2, 2, $*$
276	83 714 003 073 520	2, $*$
279	187 452 176 054 654	$*$
280	150 337 443 600 256	2, 2, 2, $*$
283	94 447 466 307 484	2, $*$
284	116 398 057 129 056	2, 2, 2, 2, $*$
287	180 238 740 202 640	2, 2, $*$
288	47 535 805 710 928	2, 2, $*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
291	50 837 867 939 808	2, $*$
292	147 004 349 275 264	2, $*$
295	413 079 425 233 376	2, 2, 2, 2, $*$
296	166 164 385 430 016	2, 2, 2, 2, $*$
299	105 836 771 424 768	2, 2, 4, $*$
300	69 869 561 232 780	6, $*$
303	127 364 844 014 196	2, $*$
304	260 104 508 754 144	2, 2, 2, $*$
307	64 921 001 378 400	2, 2, 6, $*$
308	129 706 414 328 832	2, 2, 2, 2, 2, 4, $*$
311	275 578 924 784 880	2, 2, $*$
312	55 007 593 229 520	2, 2, 2, 8
315	53 956 558 485 372	2, $*$
316	236 495 731 206 912	2, 4, $*$
319	334 649 460 889 307	$*$
320	107 604 735 927 296	2, 2, 2, 2, 2, 2, 2, 2, $*$
323	60 208 361 889 088	2, 2, 2, $*$
324	86 492 920 409 952	2, 2, 2, $*$
327	167 172 007 319 776	2, $*$
328	122 917 602 431 184	2, 2, 2, $*$
331	128 764 499 627 232	2, 2, 2, 2, $*$
332	54 534 039 728 832	2, 2, 2, 6, $*$
335	157 095 676 852 784	2, 2, $*$
336	96 632 437 523 312	4, $*$
339	50 230 652 998 720	2, $*$
340	125 166 201 149 800	2, 2, $*$
343	294 097 650 011 760	$*$
344	153 348 167 523 520	2, 2, 2, 2, 2, $*$
347	43 767 479 015 392	2, 2, 2, $*$
348	88 394 327 768 496	2, $*$
351	209 824 900 277 388	2, $*$
352	107 401 819 173 376	2, 2, 2, 2, 2, $*$
355	138 222 269 548 408	2, $*$
356	90 445 276 614 720	2, 2, 2, 2, $*$
359	319 803 844 484 736	2, 2, 2, 6, $*$
360	66 476 212 961 760	2, 2, 2, 6, $*$
363	41 345 745 280 776	2, 2, $*$
364	224 658 137 325 216	2, 2, 2, $*$
367	174 921 141 781 400	2, 2, $*$
368	65 650 695 053 472	2, 2, 2, $*$
371	113 986 401 706 720	4, $*$
372	98 613 111 826 456	2, 2, $*$
375	114 405 552 218 560	2, 2, 2, 2, 2, $*$
376	265 329 130 275 456	2, 2, $*$
379	130 340 478 179 741	$*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
380	88 676 312 034 496	2, 2, 2, 2, $*$
383	187 265 649 655 268	2, $*$
384	78 862 004 505 072	2, 2, $*$
387	46 664 825 276 208	$*$
388	146 228 546 590 656	2, 2, $*$
391	252 306 755 069 528	2, 2, $*$
392	99 179 506 891 224	2, 2, $*$
395	56 282 220 076 144	2, $*$
396	107 215 430 328 360	2, 6, $*$
399	297 453 908 465 656	2, 2, $*$
400	177 552 390 159 104	2, 2, 4, $*$
403	89 536 283 830 840	2, $*$
404	139 840 237 468 928	2, 2, 2, 2, 2, 2, $*$
407	203 486 126 093 220	$*$
408	47 224 244 599 560	2, 2, $*$
411	63 048 502 347 376	$*$
412	114 246 989 611 840	2, 2, 2, $*$
415	383 439 788 901 888	2, 2, 2, 2, 2, $*$
416	160 805 208 211 200	2, 2, 2, $*$
419	76 317 263 826 432	2, 2, 2, $*$
420	70 673 093 267 592	2, 2, $*$
423	101 355 487 027 248	2, $*$
424	139 714 422 494 496	2, 2, $*$
427	96 077 141 869 296	2, 2, $*$
428	86 059 458 814 752	2, 2, 2, $*$
431	179 226 171 447 552	2, 2, 2, 2, 4, $*$
432	79 203 439 238 016	2, 2, 2, 2, $*$
435	63 834 766 896 240	2, 2, $*$
436	168 973 350 098 160	2, 2, 2, $*$
439	430 405 842 863 776	2, $*$
440	91 591 526 067 888	2, 2, 2, $*$
443	54 402 920 073 216	2, 2, 2, 6, $*$
444	75 651 016 591 856	2, $*$
447	139 107 486 124 328	2, 2, $*$
448	16 8481 417 006 848	2, 2, 2, 4, $*$
451	110 189 736 714 112	2, 2, 2, $*$
452	85 630 565 753 632	2, 2, 2, 2, $*$
455	254 523 135 334 720	2, 2, 2, 2, $*$
456	102 985 625 465 248	2, 2, 2, $*$
459	56 716 033 878 533	$*$
460	223 099 562 322 816	2, $*$
463	376 967 304 244 236	$*$
464	129 256 158 159 872	2, 2, 2, 4, $*$
467	64 943 009 730 348	$*$
468	48 402 770 072 776	2, 2, $*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
471	189 676 969 991 920	2, 2, 2, $*$
472	70 677 745 210 048	2, 2, $*$
475	70 415 070 294 452	$*$
476	191 622 240 269 016	2, $*$
479	187 129 766 138 880	2, 2, 2, 2, $*$
480	90 539 541 900 752	2, 2, 2, $*$
483	59 036 712 169 104	2, 2, $*$
484	228 882 509 298 432	2, 6, $*$
487	214 891 568 429 544	2, $*$
488	83 236 108 564 176	2, 2, 2, $*$
491	136 797 166 469 500	2, $*$
492	70 256 977 820 352	2, 2, 2, 2, $*$
495	217 735 981 401 744	2, 2, 2, $*$
496	130 540 628 680 512	2, 2, 2, 2, 2, $*$
499	137 850 191 948 928	2, 2, 2, 2, 2, 2, $*$
500	748 87 051 645 440	2, 2, 8, $*$
503	167 652 277 005 664	2, 2, 2, $*$
504	117 753 848 946 208	4, $*$
507	43 935 964 188 092	2, $*$
508	94 722 164 702 910	$*$
511	450 083 453 461 968	2, $*$
512	121 585 751 186 432	2, 2, 2, 2, 2, $*$
515	59 190 837 146 816	2, 2, 2, 2, $*$
516	103 050 875 667 328	2, 2, 4, $*$
519	192 581 465 160 592	$*$
520	207 770 767 334 912	2, 2, 2, 2, 2, $*$
523	65 845 801 492 128	2, 2, 2, $*$
524	99 924 267 586 380	$*$
527	119 306 373 829 356	2, $*$
528	80 646 876 132 608	2, 4, 4, $*$
531	59 364 494 853 764	2, $*$
532	165 858 032 559 968	2, 2, 2, $*$
535	269 535 469 384 784	2, 2, $*$
536	139 877 154 903 264	2, 2, 2, $*$
539	108 979 682 293 104	2, 2, $*$
540	76 082 998 499 328	2, 2, 2, 2, $*$
543	102 277 335 879 216	2, 2, 2, $*$
544	170 171 346 914 400	2, 2, $*$
547	100 239 611 599 496	2, 2, $*$
548	81 146 073 342 240	2, 2, 2, $*$
551	284 018 295 384 208	2, 2, 2, $*$
552	58 145 790 841 600	2, 4, $*$
555	42 714 361 629 312	2, 2, 2, 2, $*$
556	137 111 982 884 184	$*$
559	459 504 005 806 944	4, $*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
560	104 539 288 545 888	2, 2, 2, $*$
563	40 292 119 463 296	2, 2, $*$
564	97 247 231 049 976	2, $*$
567	125 927 008 152 544	2, $*$
568	169 331 769 557 572	2, $*$
571	112 531 483 492 316	$*$
572	106 284 578 519 446	$*$
575	300 851 497 651 536	2, 2, 6, $*$
576	99 440 889 363 456	2, 2, $*$
579	64 841 640 065 634	$*$
580	162 091 897 980 416	2, 2, 2, 2, 2, $*$
583	315 712 743 760 000	2, 2, 2, $*$
584	120 781 111 243 136	2, 2, 2, 2, 2, $*$
587	41 700 539 138 784	2, $*$
588	77 136 352 277 484	2, $*$
591	188 017 011 529 408	2, 2, 2, $*$
592	120 362 096 407 488	2, 2, 4, $*$
595	123 407 652 241 200	2, $*$
596	111 423 086 216 832	2, 2, 2, $*$
599	200 587 529 329 920	2, 2, 2, 2, $*$
600	74 167 640 775 664	2, 2, $*$
603	65 189 464 743 900	2, $*$
604	210 599 362 056 600	2, 2, $*$
607	283 494 345 314 064	2, $*$
608	77 693 093 042 544	2, 2, $*$
611	63 927 734 547 712	2, 2, 2, 2, $*$
612	59 049 737 739 808	2, 2, 2, $*$
615	122 358 088 345 612	2, $*$
616	236 640 080 155 136	2, 2, 2, 2, $*$
619	167 136 181 770 768	4, $*$
620	62 929 160 433 792	2, 2, 2, 2, 6, $*$
623	234 310 551 389 880	2, 2, $*$
624	126 117 185 135 328	2, 2, 2, 2, $*$
627	49 715 215 407 074	$*$
628	137 072 214 518 016	2, 2, 2, 2, $*$
631	413 934 251 053 803	$*$
632	72 709 312 431 520	2, 2, 2, $*$
635	93 430 644 981 424	2, 2, 2, $*$
636	103 239 419 719 392	2, 2, 2, $*$
639	163 562 655 938 352	2, 2, 2, $*$
640	125 536 817 478 144	2, 2, 2, 2, 2, 2, 2, $*$
643	63 710 042 910 988	2, $*$
644	169 520 601 891 392	2, 2, 2, 2, $*$
647	186 259 420 610 054	$*$
648	42 129 448 311 216	2, 2, $*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
651	70 917 571 795 660	2, $*$
652	175 222 765 578 816	2, 2, $*$
655	208 422 777 066 492	$*$
656	171 487 870 076 616	2, $*$
659	78 006 924 364 704	2, 2, 2, $*$
660	95 922 283 503 744	2, 2, 2, 2, 2, $*$
663	156 303 223 171 116	$*$
664	141 249 175 946 830	$*$
667	86 943 900 118 880	2, 2, $*$
668	100 632 173 173 056	2, 2, 2, 2, $*$
671	236 405 463 974 624	2, 2, $*$
672	71 804 298 296 576	2, 2, 2, 2, 2, 2, $*$
675	33 786 255 080 520	2, 6, $*$
676	160 218 951 377 056	2, 2, 2, $*$
679	399 866 720 890 970	$*$
680	127 304 126 685 120	2, 2, 2, 2, $*$
683	53 268 536 743 008	2, 2, $*$
684	116 884 036 133 604	2, $*$
687	145 675 660 701 984	2, 2, $*$
688	132 399 282 017 040	2, 2, 2, $*$
691	159 739 330 259 850	$*$
692	64 145 190 788 560	2, 2, $*$
695	177 250 584 772 640	2, 2, 2, $*$
696	79 073 854 556 928	2, 2, 2, 2, 2, 2, $*$
699	61 446 923 100 352	2, 4, $*$
700	146 051 146 689 480	2, $*$
703	191 379 307 198 816	2, 2, $*$
704	135 902 634 920 640	2, 2, 2, 2, 6, $*$
707	80 654 385 057 808	2, 2, $*$
708	69 265 786 115 392	2, 2, 2, 2, 2, $*$
711	226 676 305 824 990	$*$
712	165 517 588 577 536	2, 2, 4, $*$
715	129 044 173 646 160	2, 2, $*$
716	142 884 017 667 456	2, 2, 2, 2, 2, $*$
719	295 142 954 352 064	2, 2, 2, $*$
720	66 556 194 315 520	2, 2, 2, 2, 2, 2, $*$
723	44 106 375 678 464	2, 2, $*$
724	177 577 277 236 992	2, 2, 2, 2, 2, $*$
727	237 298 925 926 464	2, $*$
728	87 720 331 935 492	2, $*$
731	52 357 378 219 896	2, 2, $*$
732	45 715 531 395 728	2, 2, 2, $*$
735	170 229 877 869 560	2, $*$
736	200 230 980 964 960	2, 2, 2, 2, $*$
739	117 267 470 903 916	2, $*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
740	155 835 540 695 904	2, 2, 2, 2, $*$
743	176 961 754 594 848	2, $*$
744	95 221 547 772 416	2, 2, 2, 2, 4, $*$
747	38 926 699 806 080	2, 2, 2, 2, $*$
748	113 451 518 102 124	2, $*$
751	392 655 655 583 232	2, 2, $*$
752	66 031 184 177 416	2, 2, $*$
755	60 044 209 599 264	2, $*$
756	122 983 851 142 936	2, 2, $*$
759	216 942 636 461 596	2, $*$
760	169 606 327 268 320	2, 2, 2, $*$
763	87 981 670 094 832	2, 2, 2, $*$
764	170 922 243 894 384	2, 2, 2, $*$
767	236 112 091 428 120	2, $*$
768	79 675 395 999 744	2, 2, 2, 2, 6, $*$
771	56 649 506 669 200	2, 2, $*$
772	116 918 279 088 456	2, $*$
775	363 392 554 481 904	2, $*$
776	95 503 251 285 696	2, 2, 2, 2, $*$
779	126 689 224 907 200	2, 2, 2, 2, $*$
780	58 626 292 809 600	2, 2, 2, 4, $*$
783	87 906 718 707 358	$*$
784	229 499 563 563 888	2, 2, $*$
787	65 687 061 224 060	2, $*$
788	75 241 588 056 832	2, 2, 2, 4, $*$
791	272 281 729 241 184	2, 2, $*$
792	96 805 771 952 624	2, 2, $*$
795	42 755 050 234 656	2, 2, $*$
796	212 295 862 606 408	$*$
799	591 229 722 686 588	$*$
800	91 133 832 953 856	2, 2, 2, 2, 2, 2, 4, 8, $*$
803	84 154 432 081 600	2, 2, 2, 2, $*$
804	70 769 050 970 240	2, 2, 4, $*$
807	116 460 527 858 576	2, 2, $*$
808	92 785 867 491 712	2, 2, $*$
811	141 176 077 670 348	$*$
812	107 880 121 117 392	2, 2, $*$
815	190 302 868 612 416	2, 2, 2, 2, 2, $*$
816	72 331 262 380 032	2, 2, 2, 16, $*$
819	62 495 609 810 360	2, $*$
820	150 129 772 412 592	2, 2, 2, $*$
823	202 269 149 884 680	2, $*$
824	173 847 641 462 984	2, 2, $*$
827	52 593 172 310 112	2, 2, 2, $*$
828	57 540 357 807 168	2, 2, $*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
831	231 430 439 474 686	$*$
832	151 735 319 302 016	2, 2, 2, 2, $*$
835	97 905 243 379 968	2, 2, 2, 2, $*$
836	106 574 556 673 328	2, 2, 2, $*$
839	189 323 891 030 236	2, $*$
840	102 672 008 545 344	2, 2, 2, 2, 2, $*$
843	36 247 864 819 752	2, 2, $*$
844	194 070 305 332 806	$*$
847	305 464 265 793 599	$*$
848	88 433 013 399 552	2, 2, 2, 2, 2, 2, 2, $*$
851	88 475 833 668 104	2, 2, $*$
852	61 001 634 157 632	2, $*$
855	219 905 112 480 260	2, $*$
856	196 628 833 193 472	2, $*$
859	131 740 792 265 832	2, 2, $*$
860	74 307 546 711 296	2, 2, 2, 2, 2, 2, 2, $*$
863	184 121 760 464 040	2, 2, $*$
864	87 010 657 721 672	2, 2, $*$
867	43 189 139 266 080	2, 2, 2, $*$
868	125 652 044 029 440	2, 2, 2, $*$
871	378 697 943 579 632	2, 2, $*$
872	72 217 296 500 544	2, 2, 2, $*$
875	76 076 627 417 280	2, 2, 2, 2, 2, $*$
876	104 266 256 409 216	4, $*$
879	132 001 994 894 808	2, $*$
880	175 919 901 181 152	2, 2, 2, 2, $*$
883	114 755 832 351 360	2, 2, 2, $*$
884	105 643 300 108 672	2, 2, 2, $*$
887	168 003 942 762 470	$*$
888	65 276 449 132 984	2, 2, $*$
891	59 649 407 701 620	2, $*$
892	135 753 327 939 042	$*$
895	220 186 329 065 848	2, 2, $*$
896	118 493 760 289 856	2, 2, 2, $*$
899	62 666 059 448 448	2, 2, 2, 2, 2, $*$
900	67 354 644 477 680	2, 2, $*$
903	154 166 750 026 848	$*$
904	165 509 583 594 896	2, 2, 2, $*$
907	59 098 223 310 816	2, 2, 2, $*$
908	92 577 319 770 096	2, 2, $*$
911	358 282 029 902 592	2, 2, $*$
912	50 159 232 095 872	2, 8, $*$
915	64 007 643 889 352	2, $*$
916	221 774 382 724 688	2, 2, 2, $*$
919	398 832 735 965 776	2, $*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
920	127 065 486 678 912	2, 2, 2, 4, $*$
923	56 723 641 011 408	2, 2, $*$
924	116 793 750 079 124	2, $*$
927	108 379 714 487 392	2, $*$
928	107 230 718 091 080	2, 2, $*$
931	196 302 919 022 284	2, $*$
932	107 162 113 774 016	2, 2, 4, $*$
935	174 801 639 671 328	2, 2, 2, $*$
936	136 249 500 795 616	2, 2, 2, $*$
939	68 120 904 567 888	$*$
940	139 911 902 508 384	2, 2, $*$
943	321 550 739 767 984	$*$
944	141 800 614 458 960	2, 2, $*$
947	48 853 498 208 640	2, 2, 2, 2, 2, 2, $*$
948	61 343 325 807 744	2, 2, $*$
951	155 798 650 043 440	2, 2, 2, $*$
952	120 487 400 462 656	2, 2, 2, $*$
955	109 296 662 847 688	2, 2, $*$
956	116 022 340 857 540	2, $*$
959	313 049 667 583 456	2, 2, 2, $*$
960	87 083 451 736 064	2, 2, 2, $*$
963	35 003 193 528 180	2, $*$
964	169 908 173 271 664	2, 2, 2, $*$
967	391 320 336 011 819	$*$
968	94 330 846 534 400	2, 2, 2, 2, 2, $*$
971	111 892 050 306 592	2, 2, 2, $*$
972	55 612 406 280 000	$*$
975	173 468 698 994 432	2, 2, 2, 2, 4, $*$
976	187 624 659 973 440	2, 2, 4, $*$
979	122 917 637 481 140	2, $*$
980	117 133 307 550 144	2, 2, 2, 2, 2, $*$
983	165 980 136 903 168	2, 2, 8, $*$
984	78 417 126 635 008	2, 2, 2, 2, $*$
987	52 890 498 780 924	$*$
988	183 492 612 938 112	2, 4, $*$
991	267 331 133 755 904	4, $*$
992	100 958 315 715 840	2, 2, 2, 2, 2, $*$
995	58 121 798 663 672	2, 2, $*$
996	94 869 810 857 984	2, 2, 2, 2, 8, $*$
999	198 667 218 432 083	$*$
1000	393 174 645 637 120	2, 2, 2, 2, 2, 2, 2, $*$
1003	343 618 331 329 368	2, $*$
1004	376 247 745 046 968	2, 2, $*$
1007	519 262 753 531 104	2, 12, $*$
1008	241 293 829 443 144	2, 2, $*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
1011	159 414 431 110 496	2, 2, $*$
1012	403 964 020 435 936	2, 2, 2, $*$
1015	850 523 653 048 024	2, $*$
1016	310 683 034 166 496	2, 2, 2, 2, $*$
1019	263 056 300 756 312	2, $*$
1020	257 685 621 628 288	2, 2, 2, $*$
1023	302 312 950 598 560	4, $*$
1024	735 791 287 062 528	2, 2, 2, 2, 2, 8, $*$
1027	321 964 647 336 270	$*$
1028	292 653 720 600 960	2, 2, 2, 2, $*$
1031	997 707 169 889 620	2, $*$
1032	167 452 848 287 872	2, 2, 2, 2, $*$
1035	192 940 902 993 664	2, 2, 2, 2, 2, 2, $*$
1036	417 122 368 239 792	2, 2, $*$
1039	1 031 598 177 475 200	4, $*$
1040	451 491 695 662 592	2, 2, 2, $*$
1043	182 230 832 448 720	2, 2, $*$
1044	282 068 118 142 440	2, $*$
1047	469 174 510 078 960	2, 2, $*$
1048	491 118 507 362 152	2, $*$
1051	422 467 670 427 588	2, $*$
1052	382 484 679 869 376	2, 2, 4, $*$
1055	860 428 955 700 480	2, 2, 2, 2, 4, $*$
1056	334 692 574 076 160	2, 2, 4, $*$
1059	184 906 353 377 866	$*$
1060	332 092 505 759 872	2, 2, 2, 2, 2, $*$
1063	870 931 465 774 552	2, 2, $*$
1064	361 342 273 701 600	2, 2, 2, $*$
1067	168 454 072 848 360	2, $*$
1068	246 030 104 311 803	$*$
1071	509 456 243 000 044	$*$
1072	344 453 432 653 728	2, 2, 2, 2, $*$
1075	387 194 724 535 152	2, 2, $*$
1076	587 291 272 307 040	2, 4, $*$
1079	917 509 239 530 448	2, 2, 2, $*$
1080	216 479 669 751 552	2, 2, 2, 2, 2, $*$
1083	153 327 501 298 524	2, $*$
1084	800 398 804 345 408	2, 2, $*$
1087	877 571 735 303 368	2, 2, $*$
1088	275 101 420 098 176	2, 2, 2, 4, $*$
1091	204 017 587 200 000	2, 2, 2, 2, 4, $*$
1092	131 364 342 236 536	2, $*$
1095	589 833 405 467 776	2, 2, 2, 4, $*$
1096	848 436 079 103 168	2, 2, 2, 2, $*$
1099	323 200 399 250 880	2, 2, 2, 2, 2, $*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
1100	216 131 869 584 576	2, 2, 4, $*$
1103	555 874 320 336 140	2, $*$
1104	269 462 331 582 768	2, 2, $*$
1107	133 156 983 237 408	2, 2, $*$
1108	471 851 869 431 208	2, 2, $*$
1111	1 087 667 980 435 056	2, 2, 2, $*$
1112	292 647 589 636 480	2, 2, 2, 2, 2, 2, $*$
1115	204 592 684 650 400	2, 4, $*$
1116	364 683 337 176 144	2, 2, $*$
1119	521 608 402 778 176	2, 2, 2, 2, $*$
1120	411 556 296 220 944	2, 2, 2, $*$
1123	253 361 077 276 784	2, 2, $*$
1124	402 779 451 680 128	2, 2, $*$
1127	605 112 385 085 240	2, $*$
1128	150 748 788 399 864	2, 2, $*$
1131	258 487 272 633 080	2, 2, $*$
1132	418 557 589 350 744	2, $*$
1135	735 219 716 431 896	2, 2, $*$
1136	381 771 476 832 240	2, 2, 2, $*$
1139	422 910 400 850 400	2, 2, $*$
1140	310 961 000 575 232	2, 2, 2, 2, 2, $*$
1143	445 528 016 453 696	2, 2, 2, 2, $*$
1144	497 603 797 141 376	2, 2, 2, $*$
1147	325 372 178 718 496	2, 2, 2, $*$
1148	190 825 693 540 736	2, 2, 2, 2, 2, $*$
1151	634 083 998 331 952	2, $*$
1152	241 750 288 005 184	2, 2, 2, $*$
1155	144 044 949 697 488	2, 2, $*$
1156	542 561 786 506 048	2, 4, $*$
1159	1 342 758 992 238 632	2, $*$
1160	388 111 763 472 192	2, 2, 6, $*$
1163	152 007 379 278 688	2, 2, $*$
1164	321 314 031 566 142	$*$
1167	591 073 494 735 032	2, 2, $*$
1168	417 025 106 647 648	2, $*$
1171	468 296 457 318 624	2, $*$
1172	273 773 849 226 240	2, 2, 2, 4, $*$
1175	539 027 520 853 944	2, 2, $*$
1176	216 991 204 834 480	2, 2, 2, $*$
1179	163 883 514 125 372	2, $*$
1180	530 171 942 487 076	2, $*$
1183	1 030 107 962 734 128	2, 6, $*$
1184	391 750 240 656 376	2, 2, $*$
1187	165 828 107 256 768	2, 2, 2, 6, $*$
1188	168 849 040 719 840	2, 2, 2, $*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
1191	468 877 484 938 068	$*$
1192	649 094 197 958 144	2, 2, $*$
1195	339 682 534 992 672	2, 2, 2, 2, $*$
1196	412 889 161 291 968	2, 2, 2, $*$
1199	971 814 685 023 624	2, 2, $*$
1200	255 993 915 269 088	2, 6, $*$
1203	107 101 492 518 744	2, $*$
1204	765 151 614 939 888	2, 2, 2, $*$
1207	624 734 778 501 380	2, $*$
1208	423 919 747 717 760	2, 2, 2, 2, 2, $*$
1211	335 686 406 049 004	2, $*$
1212	177 843 097 543 056	2, 2, 2, $*$
1215	406 274 953 313 784	2, 2, $*$
1216	647 274 719 556 480	2, 2, 2, 2, $*$
1219	361 281 434 575 824	2, $*$
1220	398 233 515 982 336	2, 2, 2, 2, 2, 2, $*$
1223	476 565 514 998 240	2, 2, $*$
1224	344 475 147 277 616	2, 2, $*$
1227	165 205 851 291 960	2, 2, $*$
1228	319 687 048 768 104	2, $*$
1231	1 450 438 541 318 144	2, 2, $*$
1232	226 960 597 533 152	2, 4, $*$
1235	193 083 191 615 744	2, 2, 2, 2, 2, $*$
1236	291 929 980 243 520	2, 2, 2, $*$
1239	445 881 976 948 128	2, 2, 2, $*$
1240	363 503 464 074 496	2, 2, 2, 2, 2, $*$
1243	297 408 024 097 000	2, $*$
1244	433 681 036 497 912	2, 2, $*$
1247	388 147 104 445 824	2, 2, 2, 2, $*$
1248	243 048 177 458 512	2, 2, 2, $*$
1251	248 974 992 099 694	$*$
1252	418 247 830 456 800	2, 2, 2, $*$
1255	1 080 367 709 801 568	2, 2, 2, $*$
1256	293 514 719 437 020	2, $*$
1259	331 785 941 376 256	2, 2, 2, 4, $*$
1260	266 526 334 359 936	2, 2, 2, 2, 6, $*$
1263	392 744 198 153 760	2, 2, $*$
1264	719 126 380 067 280	2, 2, 2, $*$
1267	188 538 566 832 208	2, 2, 2, $*$
1268	241 740 821 767 392	2, 12, $*$
1271	893 166 710 817 760	2, 2, $*$
1272	243 088 221 188 736	4, 4, $*$
1275	119 362 863 200 832	2, 2, 2, 2, $*$
1276	735 983 920 713 478	$*$
1279	1 279 777 957 452 348	2, $*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
1280	281 965 586 557 952	2, 2, 2, 2, 2, 4, $*$
1283	292 756 220 689 192	2, 2, $*$
1284	236 108 716 958 640	2, 2, 2, $*$
1287	431 513 220 530 916	6, $*$
1288	346 722 172 838 328	2, 2, $*$
1291	348 305 703 436 448	2, 2, 2, $*$
1292	294 205 712 790 480	2, 2, 2, $*$
1295	520 283 626 069 920	2, 2, 2, 2, $*$
1296	307 181 129 686 032	2, $*$
1299	286 219 724 449 434	$*$
1300	706 736 209 774 224	2, 2, 2, $*$
1303	914 811 861 359 008	2, 2, $*$
1304	480 801 576 209 760	2, 2, $*$
1307	214 736 411 672 544	2, $*$
1308	188 955 715 472 256	2, 2, 2, $*$
1311	543 871 722 461 856	2, 2, 2, $*$
1312	328 685 562 340 976	2, 2, 2, $*$
1315	297 928 787 621 664	2, 2, 2, 2, $*$
1316	507 685 280 893 744	2, 2, 2, $*$
1319	752 577 079 589 312	2, 2, 2, $*$
1320	246 232 744 242 944	2, 2, $*$
1323	87 763 443 300 766	$*$
1324	467 610 195 449 892	$*$
1327	1 141 100 756 968 560	2, $*$
1328	284 618 338 165 152	2, 2, 2, 2, $*$
1331	253 009 264 506 880	2, 2, 4, $*$
1332	230 714 236 587 392	2, 2, 4, $*$
1335	568 594 867 850 304	2, 2, 2, 2, $*$
1336	603 779 211 561 184	2, 2, $*$
1339	405 625 579 261 936	2, $*$
1340	270 625 317 246 856	2, 2, $*$
1343	688 116 555 080 384	2, 2, 2, 2, $*$
1344	264 889 187 013 760	2, 2, 2, 2, $*$
1347	140 670 518 268 492	$*$
1348	481 084 991 495 760	2, $*$
1351	1 185 670 000 362 624	2, 2, 2, 2, 2, $*$
1352	261 641 350 246 400	2, 2, 2, 2, 2, 2, 4, $*$
1355	280 683 411 905 920	2, 2, 2, 2, 2, 2, $*$
1356	306 646 466 921 943	$*$
1359	507 645 168 892 962	$*$
1360	680 041 969 283 408	2, 2, $*$
1363	341 974 168 734 080	2, 2, 4, $*$
1364	321 191 075 390 496	2, 2, 2, 2, $*$
1367	549 598 437 974 784	2, 2, 2, 2, $*$
1368	144 725 057 158 720	2, 2, 2, $*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
1371	212 057 357 748 940	2, $*$
1372	284 290 667 082 638	$*$
1375	840 094 786 588 080	2, 2, 2, $*$
1376	489 666 409 717 632	2, 2, 2, 2, $*$
1379	179 776 085 113 600	2, 2, 2, 4, $*$
1380	238 141 978 823 904	2, 2, 2, $*$
1383	402 228 195 554 816	4, $*$
1384	803 712 896 901 828	2, $*$
1387	245 567 708 326 480	2, $*$
1388	312 812 327 332 680	2, 2, $*$
1391	974 323 124 228 060	2, $*$
1392	227 901 956 319 048	2, 2, $*$
1395	219 505 394 421 976	2, 2, $*$
1396	425 057 283 308 240	2, 2, $*$
1399	1 259 316 837 469 928	2, 2, $*$
1400	245 920 233 145 856	2, 2, 2, 2, 4, $*$
1403	135 090 026 367 456	2, 2, 2, $*$
1404	445 181 783 255 704	2, 2, $*$
1407	434 187 039 383 390	$*$
1408	277 087 659 830 016	2, 2, 2, 2, 2, $*$
1411	499 277 313 615 648	2, 4, $*$
1412	331 442 729 841 600	2, 2, 2, $*$
1415	622 874 851 053 440	2, 2, 2, 2, 2, $*$
1416	330 390 482 583 440	2, 2, 2, $*$
1419	224 097 422 688 486	$*$
1420	595 037 297 461 248	2, 2, 2, 4, $*$
1423	803 533 099 018 220	2, $*$
1424	444 419 929 366 688	2, 2, 2, $*$
1427	98 504 099 939 104	2, 2, $*$
1428	239 073 691 225 760	2, 2, 2, $*$
1431	600 352 612 920 892	$*$
1432	477 113 906 666 132	2, $*$
1435	256 414 155 291 808	2, 2, $*$
1436	373 454 221 904 362	$*$
1439	1 406 499 389 805 564	$*$
1440	239 040 973 922 304	2, 2, 2, $*$
1443	103 902 575 992 349	$*$
1444	515 997 913 019 792	2, 2, $*$
1447	972 122 649 773 336	2, $*$
1448	221 487 255 369 408	2, 2, 2, 2, $*$
1451	334 870 388 843 872	2, 2, $*$
1452	228 857 363 990 568	2, $*$
1455	496 121 860 866 496	2, 2, 2, 2, $*$
1456	465 932 105 418 240	2, 2, 2, 2, 6, $*$
1459	410 631 674 657 280	4, $*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
1460	279 374 129 056 256	2, 2, 2, 2, $*$
1463	397 229 575 086 464	2, 2, $*$
1464	300 012 989 785 064	2, 2, $*$
1467	163 238 194 140 640	2, 2, $*$
1468	545 041 008 154 832	2, 2, $*$
1471	1 154 752 803 262 272	2, 2, 2, 2, $*$
1472	345 892 929 874 048	2, 2, 2, 2, 2, $*$
1475	285 364 318 395 904	2, 2, 2, 2, 2, 4, $*$
1476	207 104 130 383 232	2, 2, 2, 2, $*$
1479	772 691 945 916 996	2, $*$
1480	518 793 322 997 260	2, $*$
1483	249 993 739 340 352	2, 2, 2, $*$
1484	427 577 211 271 728	6, 6, $*$
1487	370 624 530 536 320	2, 2, 2, 2, 2, $*$
1488	192 885 199 072 304	2, 2, $*$
1491	208 828 568 248 611	$*$
1492	348 241 838 266 624	2, 2, 2, 2, $*$
1495	1 051 751 443 434 304	2, 2, 2, 2, $*$
1496	435 439 394 921 952	2, 2, 2, $*$
1499	177 692 946 829 884	2, $*$
1500	241 295 817 374 800	2, 2, 2, $*$
1503	575 325 310 824 784	$*$
1504	723 969 209 070 144	2, 2, 2, 2, $*$
1507	327 373 983 930 720	2, 4, $*$
1508	294 869 922 809 632	2, 2, 2, $*$
1511	693 765 001 390 656	2, 4, $*$
1512	196 891 105 764 020	2, $*$
1515	148 760 889 916 992	2, 2, $*$
1516	759 161 810 127 168	2, 2, 2, 2, 2, $*$
1519	1 073 843 296 188 576	2, $*$
1520	227 714 117 218 432	2, 2, 2, 2, 2, $*$
1523	265 692 492 722 592	2, 2, 2, $*$
1524	401 976 933 873 648	2, 2, 2, $*$
1527	339 209 305 288 744	$*$
1528	402 594 803 140 956	2, $*$
1531	426 729 859 474 592	2, 2, $*$
1532	241 344 760 590 848	2, 2, 2, 4, $*$
1535	889 685 155 131 744	2, 2, $*$
1536	254 427 090 101 472	2, 2, $*$
1539	175 111 418 607 744	2, 2, 2, 2, 2, $*$
1540	417 329 321 317 888	2, 2, 4, 4, $*$
1543	631 058 630 381 207	$*$
1544	433 313 952 831 312	2, 2, 2,
1547	199 774 478 719 328	4, $*$
1548	158 634 445 202 562	$*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
1551	425 922 801 877 728	2, $*$
1552	465 091 523 666 688	2, $*$
1555	263 373 884 711 424	2, 2, 4, $*$
1556	494 862 648 774 240	2, 2, 2, $*$
1559	851 826 764 826 128	2, 2, 2, $*$
1560	329 828 242 843 712	2, 2, 2, 2, $*$
1563	171 635 428 683 432	6, $*$
1564	514 524 108 990 214	$*$
1567	742 759 261 305 440	2, 2, $*$
1568	238 668 004 122 624	2, 2, 2, 2, 2, 4, $*$
1571	249 991 041 690 468	2, $*$
1572	249 725 496 862 976	2, 2, 2, 2, 2, $*$
1575	457 845 752 762 820	2, $*$
1576	481 972 403 521 856	2, 2, $*$
1579	478 215 151 579 240	2, 2, $*$
1580	350 580 141 520 320	2, 2, 2, 2, $*$
1583	500 841 117 364 288	2, 2, 2, 2, $*$
1584	361 674 674 835 708	2, $*$
1587	123 098 915 431 308	2, $*$
1588	432 280 473 897 360	2, 2, $*$
1591	1 519 409 568 101 184	2, 2, $*$
1592	243 804 340 628 160	2, 4, $*$
1595	208 302 366 779 760	2, 2, 2, $*$
1596	258 157 005 318 836	2, $*$
1599	587 601 311 166 200	$*$
1600	613 906 621 696 000	2, 2, 2, 4, $*$
1603	227 593 532 065 020	2, $*$
1604	343 979 972 448 976	2, 2, 2, $*$
1607	887 881 941 311 616	2, 2, 2, $*$
1608	216 755 488 697 952	2, 2, 2, $*$
1611	242 115 291 327 872	2, 2, 4, $*$
1612	453 997 034 180 916	$*$
1615	1 346 727 267 581 176	2, 2, $*$
1616	431 428 146 684 800	2, 2, 2, 2, $*$
1619	208 158 355 148 520	2, $*$
1620	220 630 401 004 800	2, 2, 2, 2, $*$
1623	585 105 258 080 016	2, 2, $*$
1624	526 819 770 570 912	2, 2, 2, $*$
1627	260 925 880 728 832	2, 2, $*$
1628	316 485 233 699 152	2, 2, $*$
1631	573 475 690 199 824	2, 2, $*$
1632	133 581 319 275 316	2, $*$
1635	201 053 672 282 496	2, 8, $*$
1636	778 455 427 082 520	2, 2, $*$
1639	1 027 290 804 724 700	2, $*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
1640	450 439 726 268 160	2, 2, 2, 2, 2, 6, $*$
1643	169 473 591 456 360	2, $*$
1644	239 583 303 385 968	2, $*$
1647	419 080 379 377 584	2, 2, $*$
1648	417 541 394 244 736	2, 2, 2, 4, $*$
1651	348 183 101 172 848	2, $*$
1652	176 490 057 293 616	2, 2, 2, $*$
1655	497 710 087 941 888	2, 2, 2, 2, 2, $*$
1656	368 702 392 445 528	2, 2, $*$
1659	234 415 731 916 020	$*$
1660	504 677 353 578 080	2, 2, 2, $*$
1663	808 223 937 859 576	2, $*$
1664	572 233 445 669 376	2, 2, 2, 2, 2, 2, 2, $*$
1667	194 418 630 891 460	$*$
1668	266 742 914 592 832	2, $*$
1671	653 176 377 343 624	2, $*$
1672	324 045 421 313 136	2, 2, $*$
1675	452 051 481 049 056	2, 4, $*$
1676	384 046 574 032 608	2, 2, 2, 2, $*$
1679	1 080 091 122 365 952	2, 2, 2, 2, 2, $*$
1680	186 627 000 247 296	2, 2, 4, 4, $*$
1683	96 167 065 974 211	$*$
1684	806 704 168 791 040	2, 2, 2, 4, $*$
1687	551 851 852 268 352	2, 2, $*$
1688	293 193 537 081 648	2, $*$
1691	245 407 988 775 288	2, $*$
1692	293 449 196 214 512	2, 2, $*$
1695	365 633 883 140 640	2, 2, $*$
1696	746 407 032 649 920	2, 2, 4, $*$
1699	552 306 717 148 533	$*$
1700	347 284 669 135 680	2, 2, 2, 2, 6, $*$
1703	623 186 395 934 304	2, 2, 2, $*$
1704	219 323 859 906 496	2, 2, 2, 2, $*$
1707	123 403 093 879 176	2, 2, $*$
1708	254 945 353 196 841	$*$
1711	1 477 730 592 073 368	2, $*$
1712	346 992 038 425 152	6, $*$
1715	168 863 073 697 472	2, 2, 2, 2, 2, $*$
1716	239 043 605 657 472	2, 2, 2, 2, $*$
1719	645 024 919 816 308	2, $*$
1720	440 027 482 907 520	2, 2, 2, $*$
1723	205 702 456 991 952	2, $*$
1724	431 098 148 154 096	2, $*$
1727	629 943 894 611 136	2, 2, 2, 6, $*$
1728	262 210 915 791 360	2, 2, 2, 2, 2, 6, 6, $*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
1731	232 530 415 564 160	2, 2, 4, $*$
1732	339 094 158 293 824	4, $*$
1735	847 689 497 920 384	2, 2, 2, 2, 2, 2, $*$
1736	329 802 308 298 240	2, 2, 2, 2, 2, $*$
1739	192 807 792 558 704	2, 2, $*$
1740	289 659 657 781 584	2, 2, $*$
1743	360 426 682 957 488	2, 2, 2, $*$
1744	672 556 006 837 632	2, 2, 2, 2, $*$
1747	311 471 571 966 720	2, 2, $*$
1748	254 574 419 417 120	2, 2, $*$
1751	822 444 151 766 088	2, 2, $*$
1752	237 142 490 393 632	2, 4, $*$
1755	227 309 677 137 424	2, 2, 2, $*$
1756	742 352 400 685 568	2, 2, $*$
1759	1 567 138 640 970 576	2, 2, $*$
1760	267 511 171 837 952	2, 2, 2, 2, 2, 2, $*$
1763	191 226 748 847 688	2, $*$
1764	242 620 192 181 696	2, 2, 2, $*$
1767	366 231 958 659 740	$*$
1768	537 519 539 868 024	2, 2, $*$
1771	267 978 025 993 600	2, 2, 2, 2, $*$
1772	266 503 688 367 636	2, $*$
1775	634 032 514 967 328	2, 2, 2, $*$
1776	468 724 605 214 288	2, 2, $*$
1779	197 041 343 860 288	2, 4, $*$
1780	561 635 162 076 880	2, 2, 2, $*$
1783	1 191 491 542 848 845	$*$
1784	319 892 848 504 704	2, 2, 2, 2, 2, 6, $*$
1787	194 450 854 821 408	2, $*$
1788	155 961 829 163 940	2, $*$
1791	602 110 096 648 896	2, $*$
1792	395 421 392 535 552	2, 2, 2, 2, 8, $*$
1795	241 104 008 702 544	2, 2, $*$
1796	375 199 188 738 912	2, 2, 2, $*$
1799	635 809 874 263 522	$*$
1800	188 125 965 043 152	2, 2, $*$
1803	199 867 858 649 760	2, 2, 2, $*$
1804	518 513 826 873 192	2, $*$
1807	753 752 534 030 368	$*$
1808	295 549 176 498 048	2, 2, 2, 2, $*$
1811	373 902 933 687 168	2, 2, 2, 2, $*$
1812	200 145 486 858 144	2, 2, 2, 2, $*$
1815	584 680 503 195 904	2, 2, 2, 2, 4, $*$
1816	512 136 417 149 112	2, 2, $*$
1819	435 822 576 736 192	2, 2, 2, 2, $*$

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
1820	439 275 265 071 616	2, 2, *
1823	454 699 165 301 124	2, *
1824	369 126 858 638 240	2, 4, *
1827	137 171 011 387 188	2, *
1828	341 113 087 365 152	2, 2, 2, 2, *
1831	1 401 753 458 935 488	2, 2, 2, 2, *
1832	384 740 261 596 960	2, 2, *
1835	222 800 075 360 928	2, 2, 2, 6, *
1836	475 207 534 897 344	2, 2, 2, 6, *
1839	657 619 109 824 768	2, *
1840	375 677 242 235 904	2, 2, 2, 2, *
1843	349 899 041 174 800	2, *
1844	470 547 062 332 416	2, 2, 2, 2, 8, *
1847	487 930 442 984 736	2, 2, 2, *
1848	163 862 844 209 760	2, 2, 2, 2, *
1851	133 326 881 841 984	2, 2, 4, *
1852	393 269 512 730 616	*
1855	898 903 682 576 960	2, 2, 2, 2, 2, *
1856	279 933 999 065 104	2, 2, *
1859	278 340 505 613 568	2, 2, 2, 2, 2, 6, *
1860	272 766 253 747 568	2, 2, *
1863	305 154 737 951 658	*
1864	653 721 695 843 520	2, 2, *
1867	273 530 397 873 832	2, 2, *
1868	310 224 620 717 568	2, 2, 2, 4, *
1871	1 137 511 663 237 312	2, 2, 2, *
1872	157 349 645 057 280	2, 2, 2, 2, 2, *
1875	181 246 706 491 200	2, 2, *
1876	581 080 599 675 210	*
1879	1 135 355 021 751 328	4, *
1880	425 458 031 734 272	2, 2, 2, 2, 2, *
1883	178 794 531 492 096	2, *
1884	213 821 214 471 792	2, 2, *
1887	647 853 000 663 474	*
1888	653 079 070 581 152	2, 2, 2, 2, *
1891	265 591 505 291 734	*
1892	249 181 622 550 784	2, 2, 2, 2, 2, 2, *
1895	718 222 058 062 136	2, 2, *
1896	340 314 433 505 280	2, 2, 2, 2, 2, 4, *
1899	173 664 944 165 620	*
1900	406 355 255 363 376	2, 2, *
1903	624 741 014 341 504	2, 2, 2, 2, 2, *
1904	332 974 917 780 992	2, 2, 4, *
1907	151 250 134 344 324	2, *
1908	249 460 795 339 820	2, *

d	$h(\Delta)$	Struktur von $Cl(\Delta)$
1911	577 247 883 154 628	$*$
1912	369 408 025 707 232	2, 2, 2, $*$
1915	346 400 380 164 088	2, $*$
1916	409 072 660 637 796	6, $*$
1919	798 909 332 143 888	2, $*$
1920	296 153 361 023 488	2, 2, 2, 2, 2, 4, $*$
1923	141 984 845 816 100	2, $*$
1924	791 298 981 966 816	2, 6, $*$
1927	931 268 713 381 496	2, 2, $*$
1928	200 313 406 919 728	2, 2, 2, $*$
1931	313 939 013 681 600	2, 2, 2, 2, 2, $*$
1932	186 658 109 377 224	2, 2, $*$
1935	442 018 653 548 960	2, 2, 2, $*$
1936	600 657 593 753 600	2, 2, 8, $*$
1939	388 393 325 513 200	2, 2, 2, $*$
1940	231 959 377 124 288	2, 2, 2, 2, $*$
1943	634 319 333 341 056	2, 2, $*$
1944	364 687 673 946 588	6, $*$
1947	80 875 188 624 312	2, 2, $*$
1948	541 899 431 583 714	$*$
1951	1 275 143 003 658 236	2, $*$
1952	305 909 566 091 776	2, 2, 2, 2, 2, 2, 2, $*$
1955	207 118 211 930 880	2, 2, 4, $*$
1956	258 267 768 771 288	2, 2, $*$
1959	603 502 032 345 440	2, 2, 2, $*$
1960	363 818 515 594 240	2, 2, 2, $*$
1963	430 080 752 609 590	$*$
1964	537 312 971 711 664	2, 2, $*$
1967	466 120 551 391 208	2, 2, $*$
1968	205 994 061 677 952	2, 2, 2, 2, 6, $*$
1971	224 654 306 952 768	4, $*$
1972	515 403 065 278 520	2, 2, $*$
1975	960 854 930 519 168	2, 2, 4, $*$
1976	461 806 596 990 304	2, 2, 2, $*$
1979	368 082 004 222 232	2, $*$
1980	164 290 335 597 760	2, 2, 2, $*$
1983	440 638 587 767 680	2, 2, 2, 2, 2, $*$
1984	417 354 607 997 600	2, 2, $*$
1987	414 583 075 493 664	4, $*$
1988	2134 30 779 006 144	2, 2, 2, 2, 2, $*$
1991	610 977 009 691 480	2, 2, $*$
1992	2475 89 951 923 200	2, 2, $*$
1995	117 827 880 234 080	4, $*$
1996	585 737 880 509 088	4, $*$
1999	1 246 077 121 417 780	$*$

Literaturverzeichnis

[1] A.O.L. Atkin, *NUDUPL, NUCOMP*, Manuskript, 1988.

[2] E. Bach, *Explicit bounds for primality testing and related problems*, Math. Comp. **55** (1990), 355–380.

[3] G.H. Bradley, *Algorithms for Hermite and Smith normal form matrices and linear diophantine equations*, Math. Comp. **25** (1971), 897–907.

[4] J. Buchmann, M. Diehl, R.Roth, *LIPS - A system for distributed applications*, preprint 1991.

[5] J. Buchmann, S. Düllmann, *A probabilistic class group and regulator algorithm and its implementation*, Computational Number Theory (A. Pethö, M. Pohst, H.C. Williams, H.G. Zimmer, eds.), Walter de Gruiter Verlag Berlin, 1991, 53-72.

[6] J. Buchmann, S. Düllmann, *Distributed class group computation*, Festschrift aus Anlaß des sechzigsten Geburtstages von Herrn Prof. Dr. G. Hotz, Universität des Saarlandes, 1991.

[7] J. Buchmann, S. Düllmann und H.C. Williams, *On the complexity and efficiency of a new key exchange system*, Advances in Cryptology - Proceedings EUROCRYPT'89 (J. Quisquater, J. Vandewalle, eds.), Lecture Notes in Computer Science **434**, Springer Verlag Berlin, 1990, 597–616.

[8] J. Buchmann und H.C. Williams, *A key exchange system based on imaginary quadratic fields*, J. Cryptology **1** (1988), 107–118.

[9] J. Buchmann und H.C. Williams, *On the computation of the class number of an algebraic number field*, Math. Comp. **53** (1989), 679–688.

[10] D.A. Buell, *Class groups of quadratic fields*, Math. Comp. **30** (1976), 610–623.

[11] D.A. Buell, *Binary quadratic forms: classical theory and modern computations*, Springer Verlag New York, 1989.

[12] H. Cohen und H.W. Lenstra Jr., *Heuristics on class groups of number fields*, Number Theory (Noordwijkerhout 1983), Lecture Notes in Math. **1068** (1984), 33–62.

[13] H. Cohen, *A course in computational algebraic number theory*, Buch-Manuskript, 1991.

[14] W. Diffie und M. Hellman, *New directions in cryptography*, IEEE Transactions on Information Theory **22** (1976), 472–492.

[15] P.D. Domich, R. Kannan and L.E. Trotter, *Hermite normal form computation using modulo determinant arithmetic*, Math. Op. Res. **12** (1987), 50–59.

[16] S. Düllmann, *Ein neues Verfahren zum öffentlichen Schlüsselaustausch*, Staatsexamensarbeit, Universität Düsseldorf, 1988.

[17] C.F. Gauss, *Disquisitiones Arithmeticae*, Fleischer, Leipzig, 1801.

[18] J.L. Hafner and K.S. McCurley, *A rigorous subexponential algorithm for computation of class groups*, Journal AMS, to appear.

[19] H. Heuser, *Lehrbuch der Analysis, Teil 1*, Teubner Verlag Stuttgart, 1980.

[20] Hua Loo Keng, *Introduction to Number Theory*, Springer Verlag Berlin, 1982.

[21] T.W. Hungerford, *Algebra*, Springer Verlag New York, 1974.

[22] R. Kannan and A. Bachem, *Polynomial algorithms for computing the Smith and Hermite normal form of an integer matrix*, SIAM J. Comput. **8** (1979), 499–507.

[23] D.E. Knuth, *The art of computer programming, Vol. 2: Seminumerical algorithms*, Addison-Wesley, Reading, Massachusets, 2. Auflage, 1981.

[24] N. Koblitz, *A course in number Theory and cryptography*, Springer Verlag New York, 1987.

[25] E. Kranakis, *Primality and Cryptography*, Teubner Verlag Stuttgart, 1986.

[26] J.C. Lagarias, *Worst-case complexity bounds for algorithms in the theory of integral quadratic forms*, J. Algorithms **1** (1980), 142–186.

[27] B.A. LaMacchia and A.M. Odlyzko, *Solving large sparse linear systems over finite fields*, Advances in Cryptology - Proceedings Crypto'90 (A.J. Menezes, S.A. Vanstone, eds.), Lecture Notes in Computer Science **537**, Springer Verlag Berlin, 1991, 109–133.

[28] H.W. Lenstra Jr, *On the calculation of regulators and class numbers of quadratic fields*, London Math. Soc. Lecture Notes **56** (1982), 123–150.

[29] K.S. McCurley, *Cryptographic key distribution and computation in class groups*, Number Theory and applications (R.A. Mollin, ed.), NATO ASI series, Series C, Vol. 265, Dordrecht 1989, 459–479.

[30] W. Narkiewicz, *Elementary and analytic theory of algebraic numbers*, Warszawa 1974.

[31] I. Nassi, B. Shneiderman, *Flowchart techniques for structured programming*, SIGPLAN Notices **8** (1973), No. 8, 12–26.

[32] A.M. Odlyzko, *Discrete logarithms in finite fields and their cryptographic significance*, Advances in cryptology: Proceedings EUROCRYPT'84, Lecture Notes in Computer Science **209**, Springer Verlag Berlin, 1985, 224–314.

[33] C. Pomerance, *Analysis and Comparison of some integer factoring algorithms*, Computational Methods in Number Theory (R. Tijdeman and H.W. Lenstra Jr., eds.), Mathematisch Centrum Tract 154, Amsterdam, 1982, 89–139.

[34] C.P. Schnoor, *Refined analysis and improvements on some factoring algorithms*, J. Algorithms **2** (1982), 101–127.

[35] A. Schönhage and V. Strassen, *Schnelle Multiplikation großer Zahlen*, Computing **7** (1971), 281-292.

[36] A. Schönhage, *Schnelle Berechnung von Kettenbruchentwicklungen*, Acta Informatica **1** (1971), 139–144.

[37] R. Schoof, *Quadratic fields and factorization*, Computational Methods in Number Theory (R. Tijdeman and H.W. Lenstra Jr, eds.), Mathematisch Centrum Tract 154, Amsterdam 1982, 235–286.

[38] M. Seysen, *A probabilistic factorization algorithm with quadratic forms of negative discriminant*, Math. Comp. **48** (1987), 737–780.

[39] D. Shanks, *Class number, a theory of factorization, and genera*, Proc. Symposium Pure Mathematics, American Mathematical Society **20** (1971), 415–440.

[40] D. Shanks, *Five number-theoretic algorithms*, Proc. Second Manitoba Conference on Numerical Mathematics (1972), 51–70.

[41] D. Shanks, *On Gauss and Composition*, Teil I und II, Number Theory and applications (R.A. Mollin, ed.), NATO ASI series, Series C, Vol. 265, Dordrecht 1989, 163–178 und 179–204.

[42] R.D. Silverman, *The multiple polynomial quadratic sieve*, Math. Comp. **48** (1987), 329–339.

[43] B.L. van der Waerden, *Algebra I*, Springer Verlag Berlin, 8. Auflage, 1971.

[44] B.L. van der Waerden, *Algebra II*, Springer Verlag Berlin, 5. Auflage, 1967.