Der LLL-Algorithmus und seine Anwendung auf Faktorisierung ganzzahliger Polynome

von
Benedikt Annweiler

Seminar in Computeralgebra

angefertigt am Lehrstuhl D für Mathematik
Prof. Dr. Gabriele Nebe

Inhaltsverzeichnis

1 Einleitung

Die Ausarbeitung dieses Seminars basiert auf dem Artikel *Factoring Polynomials with Rational Coefficients* [LLL82], veröffentlicht 1982 von Arjen Lenstra, Hendrik Lenstra und Laszlo Lovasz. Der Vortrag beschäftigt sich insbesondere mit dem LLL-Algorithmus, welcher von jenen drei Autoren im selbigen Jahr entwickelt wurde. Dieser Algorithmus errechnet in polynomieller Laufzeit ($O(d^5 n \log^3 B)$, L Gitter in $\mathbb{R}^n$, $d = Rang(L)$, $B = max\{|b| : b \in L \; Basisvektor\}$) aus einer beliebigen Gitterbasis eine Gitterbasis, welche aus sehr kurzen, fast orthogonalen Gittervektoren besteht. Dafür werden wir eine sogenannte LLL-reduzierte Gitterbasis einführen.

Im ersten Teil dieser Ausarbeitung wird der LLL-Algorithmus beschrieben und wie man mit Hilfe dessen eine sogenannte LLL-reduzierte Gitterbasis bestimmen kann. Weiterhin kann man den LLL-Algorithmus so transformieren, dass er nur mit Gram-Matrizen arbeitet.

Der zweite Teil der Ausarbeitung beschäftigt sich mit dem Zusammenhang zwischen Polynomen, deren Faktoren und Gittern, sowie mit der Anwendung des Algorithmus auf ein Polynom. Wir werden sehen, wie wir aus einem irreduziblen Faktor $h_0 \; mod \; p^k$ des Polynoms $f \in \mathbb{Z}[X]$ mit Hilfe des im ersten Teil vorgestellten LLL-Algorithmus einen ganzzahligen irreduziblen Faktor berechnen.

Im abschließenden dritten Teil wird der eigentliche Algorithmus zur Polynomfaktorisierung erläutert. Wir bedienen uns dabei verschiedener Faktorisierungsalgortihmen. Wir werden eine geeignete kleine Primzahl p und einen p-adischen irreduziblen Faktor von f finden. Dies werden wir mit dem Berlekampalgorithmus über $\mathbb{F}_p$, kombiniert mit dem Henselschen Lemma erreichen können. Anschließend suchen wir einen irreduziblen Faktor h_0 von f in $\mathbb{Z}[X]$, für den $h \mid h_0$ gilt. Das bedeutet, dass h_0 zu einem passenden Gitter gehört, wobei die Aussage $h_0 \mid f$ impliziert, dass die Koeffizienten von h_0 relativ klein sind. Der LLL-Algorithmus liefert uns relativ kurze Vektoren eines solchen Gitters. Mit der wiederhohlten Anwendung des Algorithmus lassen sich so alle irreduziblen Faktoren bestimmen.

2 Grundvoraussetzungen

In der gesamten Ausarbeitung dieses Seminars betrachten wir den Euklidischen Vektorraum $\mathbb{R}^n$ und die darauf definierte symmetrische Bilinearform $\Phi : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$, das Standardskalarprodukt $(x, y) \mapsto \sum_i x_i y_i$.
Es sei $f \in \mathbb{Z}[X]$ das zu faktorisierende Polynom vom Grad n. Für ein Polynom $\sum_i a_i X^i$ sei

$$\left| \sum_i a_i X^i \right| = (\sum_i a_i^2)^{1/2}.$$

Desweiteren bezeichne $|v|$ für einen Vektor v die euklidische Länge des Vektors.
Mit μ_{ij} und b_i^* werden die Skalare bzw. Vektoren des Gram-Schmidt-Orthogonalisierungsverfahrens, definiert in (3.2), bezeichnet.

3 Der LLL-Algorithmus

Der LLL-Algorithmus arbeitet in seiner ursprünglichen Form mit Gittern und lässt sich so umschreiben, dass er auschließlich mit Gram-Matrizen arbeitet. Wir werden am Anfang dieses Kapitels eine LLL-reduzierte Basis definieren und darauf aufbauend den LLL-Algorithmus angeben.

Definition 3.1.
Sei $n \in \mathbb{N}$. Eine Teilmenge $L \subset \mathbb{R}^n$ heißt *Gitter*, falls eine Basis $b_1, b_2, ..., b_n$ des $\mathbb{R}^n$ existiert mit

$$L = \sum_{i=1}^{n} \mathbb{Z} b_i = \left\{ \sum_{i=1}^{n} r_i b_i : r_i \in \mathbb{Z},\ 1 \leq i \leq n \right\}$$

Dann heißt $b_1, b_2, ..., b_n$ eine *Basis* von L.
Die *Determinante* von L ist definiert als

$$d(L) = |det(b_1, b_2, ..., b_n)|$$

wobei die b_i als Spaltenvektoren dargestellt werden. $d(L) \in \mathbb{R}_+$ ist unabhängig von der Basiswahl.

Für den später aufgeführten LLL-Algorithmus brauchen wir noch das bekannte Gram-Schmidt-Orthogonalisierungsverfahren.

Satz 3.2. (Gram-Schmidt-Orthogonalisierungsverfahren)
Eingabe: Eine Basis $(b_1, b_2, ..., b_n)$ des $\mathbb{R}^n$.
Ausgabe: Eine Orthogonalbasis $B^* := (b_1^*, b_2^*, ..., b_n^*)$ des $\mathbb{R}^n$
Algorithmus: Für $i = 1, ..., n$ berechne

$$b_i^* := b_i - \sum_{j=1}^{i-1} \mu_{i,j} b_j^*$$

mit $\mu_{i,j} = \frac{\Phi(b_i, b_j^*)}{\Phi(b_j^*, b_j^*)}$.

b_i^* ist die orthogonale Projektion von b_i auf $span\{\mathbb{R}b_1, ..., \mathbb{R}b_{i-1}\}$. Weiter bekommen wir mir $\Phi(b_j^*, b_j^*) \leq \Phi(b_j, b_j)$ die Hadamard Ungleichung:

Bemerkung 3.3. (Hadamard-Ungleichung)
Sei $b_1, b_2, ..., b_n$ eine Basis des Gitters L, so gilt:

$$det(L) \leq \prod_{j=1}^{n} |b_j|$$

Beweis.
Es gilt $det(L) = \prod_{j=1}^{n} \Phi(b_j^*, b_j^*)$. Mit $\Phi(b_j^*, b_j^*) \leq \Phi(b_j, b_j)$ und $|b_j| = \Phi(b_j, b_j)$ folgt die Behauptung. □

Definition 3.4.
Eine Basis $b_1, b_2, ..., b_n$ heißt *LLL-reduziert*, falls

$$(i) \ |\mu_{i,j}| \leq \frac{1}{2} \text{ für } 1 \leq j < i \leq n$$

$$(ii) \ |b_i^* + \mu_{i,i-1} b_{i-1}^*|^2 \geq \frac{3}{4} |b_{i-1}^*|^2 \text{ für } 1 < i \leq n$$

gilt.

Die Beschränkung $\mu_{i,j}$ sind eine Mindesanforderung an den Grad der Orthogonalität der Gitterverktoren. Falls $\mu_{ij} = 0$ so stehen die Gittervektoren mit den Indizes i und j senkrecht aufeinander. Die Forderung (ii) stellt sicher, dass zwei Vektoren in ihrer Länge nicht zu weit auseinander liegen.

Proposition 3.5.
Sei $b_1, b_2, ..., b_n$ eine LLL-reduzierte Basis für ein Gitter L in $\mathbb{R}^n$ und sei $b_1^*, b_2^*, ..., b_n^*$ definiert als die orthogonale Basis aus (3.2). Dann gilt:

$$(i) \ \ |b_j|^2 \leq 2^{i-1} \cdot |b_i^*|^2 \text{ für } 1 \leq j \leq i \leq n,$$
$$(ii) \ \ d(L) \leq \prod_{i=1}^{n} |b_i| \leq 2^{n(n-1)/4} \cdot d(L),$$
$$(iii) \ |b_1| \leq 2^{(n-1)/4} \cdot d(L)^{1/n}.$$

Beweis.
zu (i): Aus der Definition von LLL-reduziert folgt

$$|b_i^*|^2 \geq (\frac{3}{4} - \mu_{i,i-1}^2) \cdot |b_{i-1}^*|^2 \geq \frac{1}{2} \cdot |b_{i-1}^*|^2 \text{ für } 1 < i \leq n.$$

Mit Induktion erhalten wir

$$|b_j^*|^2 \leq 2^{i-j} \cdot |b_i^*|^2 \text{ für } 1 \leq j \leq i \leq n$$

Aus dem Gram-Schmidt-Orthogonalisierungsverfahren und (3.4)(i) folgt

$$\begin{aligned}|b_i|^2 &= |b_i^*|^2 + \sum_{j=1}^{i-1} \mu_{ij}^2 \left|b_j^*\right|^2 \\ &\leq |b_i^*|^2 + \sum_{j=1}^{i-1} \frac{1}{4} 2^{i-j} |b_i^*|^2 \\ &= (1 + \frac{1}{4}(2^i - 2)) \cdot |b_i^*|^2 \\ &\leq 2^{i-1} \cdot |b_i^*|^2 .\end{aligned}$$

Es folgt also, dass

$$\left|b_j\right|^2 \leq 2^{j-1} \cdot \left|b_j^*\right|^2 \leq 2^{i-1} \cdot |b_i^*|^2$$

für alle $1 \leq j \leq i \leq n$ gilt.
Zu (ii): Aus dem Gram-Schmidt-Orthogonalisierungsverfahren und der Basisunabhängigkeit der Gitterdeterminante folgt

$$d(L) = |det(b_1^*, b_2^*, ..., b_n^*)|$$

und mit den paarweise orthogonalen b_i^*

$$d(L) = \prod_{1=1}^{n} |b_i^*| .$$

Mittels $\left|b_i^*\right| \leq |b_i|$ und $|b_i| \leq 2^{(i-1)/2} \cdot \left|b_i^*\right|$ folgt schließlich

$$d(L) \leq \prod_{i=1}^{n} |b_i| \leq 2^{n(n-1)/4} \cdot d(L).$$

Zu (iii): Es gilt für (i) mit $j = 1$

$$|b_1|^2 \leq 2^{i-1} \cdot |b_1^*|^2 .$$

Das Produkt über $i = 1, 2, ..., n$ ergibt dann

$$|b_1| \leq 2^{(n-1)/4} \cdot d(L)^{1/n}.$$ □

Proposition 3.6.
Sei $L \subset \mathbb{R}^n$ ein Gitter mit LLL-reduzierter Basis $b_1, b_2, ..., b_n$. Dann gilt

$$|b_1|^2 \leq 2^{n-1} \cdot |x|^2$$

für alle $x \in L$, $x \neq 0$.

Beweis.
Sei $x = \sum_{i=1}^{n} a_i b_i = \sum_{i=1}^{n} \alpha_i b_i^*$ mit $a_i \in \mathbb{Z},\ \alpha_i \in \mathbb{R}$ für alle $1 \leq i \leq n$. Sei i maximal mit $a_i \neq 0$, so ist $\alpha_i = a_i$ und

$$|x|^2 \geq \alpha_i^2 \cdot |b_i^*|^2 \geq |b_i^*|^2 \geq (\frac{1}{2})^{i-1} |b_1| \geq (\frac{1}{2})^{n-1} |b_1| .$$

Dies zeigt die Behauptung. □

Proposition 3.7.
Sei $L \subset \mathbb{R}^n$ ein Gitter mit LLL-reduzierter Basis $b_1, b_2, ...b_n$. Seien $x_1, x_2, ..., x_t \subset L$ linear unabhängig. Dann gilt

$$\left|b_j\right|^2 \leq 2^{n-1} \cdot max \left\{ |x_1|^2 , |x_2|^2 , ..., |x_t|^2 \right\}$$

für $j = 1, 2, ..., t$.

Beweis.
Sei $x_j = \sum_{i=1}^{n} a_{ij} b_i$ mit $a_{ij} \in \mathbb{Z}$ mit $1 \leq i \leq n$ für $1 \leq j \leq n$. Für ein festes j sei $i(j)$ das größte i mit $a_{ij} \neq 0$. Dann erhalten wir mit Proposition (3.6)(Beweis)

$$\left|x_j\right|^2 \geq \left|b_{i(j)}^*\right|^2$$

für $1 \leq j \leq n$. Ordne die x_j neu, so dass $i(1) \leq i(2) \leq ... \leq i(t)$. Es ist $j \leq i(j)$ für $1 \leq j \leq t$, sonst könnten $x_1, x_2, ..., x_j$ mit $\sum_{k=1}^{j-1} \mathbb{R} b_k$ dargestellt werden, welches im Widerspruch zu der linearen Unabhängigkeit von $x_1, x_2, ...x_n$ steht.
Also erhalten wir aus $j \leq i(j)$ und (3.5)(i):

$$\left|b_j\right|^2 \leq 2^{i(j)-1} \cdot \left|b_{i(j)}^*\right|^2 \leq 2^{n-1} \cdot \left|b_{i(j)}^*\right|^2 \leq 2^{n-1} \cdot \left|x_j\right|^2$$

für $j = 1, 2, ..., t$. □

Satz 3.8.
Jedes Gitter $L \subset \mathbb{R}^n$ hat eine LLL-reduzierte Basis.

Zum Beweis geben wir den LLL-Algorithmus an:

Im Algorithmus iterieren wir über k. Beim jeweils aktuellen Laufindex k ist $(b_1, b_2, ..., b_{k-1}) \subset L$ eine LLL-reduzierte Basis. Erreicht der Laufindex k den Wert $n+1$, so haben wir eine LLL-reduzierte Basis von L gefunden.

Algorithmus 3.9. (LLL-Reduktion)
Eingabe: Gitterbasis $(b'_1, b'_2, ..., b'_n)$ von L.
Ausgabe: LLL-reduzierte Gitterbasis $(b_1, b_2, ..., b_n)$ von L und Transformationsmatrix $T = (t_{ij}) \in GL_n(\mathbb{Z})$ mit $b_i = \sum_{j=1}^{n} t_{ij} b'_j$.
Algorithmus: $T := I_n$; $b_i := b'_i$ für $1 \leq i \leq n$; $k := 2$
while $(k \leq n)$ do
  if $(k \geq 2)$ then $\mu_{k,k-1} := \Phi(b_k, b^*_{k-1}) / \Phi(b^*_{k-1}, b^*_{k-1})$;
    if $(|\mu_{k,k-1}| \geq 1/2)$ then $r := round(\mu_{k,k-1})$;
      $b_k := b_k - r b_{k-1}$; $T_k := T_k - r T_{k-1}$;
      for $j = 1, ..., k-2$ do
        $\mu_{k,j} = \mu_{k,j} - r\mu_{k-1,j}$;
      end for;
      $\mu_{k,l} = \mu_{k,l} - r$;
    end if;
    for $j = 1, ..., k-2$ do
      $\mu_{k,j} := \Phi(b_k, b^*_j) / \Phi(b^*_j, b^*_j)$;
    end for;
    $b^*_k := b_k - \sum_{j=1}^{k-1} \mu_{k,j} b^*_j$;
    if $(\left|b^*_k + \mu_{k,k-1} b^*_{k-1}\right|^2 \geq \frac{3}{4} \left|b^*_{k-1}\right|^2)$ then *(Fall 2)*
      for $j = k-2, ..., 1$ do $r := round(\mu_{k,j})$;
        $b_k := b_k - r b_j$; $T_k := T_k - T_j$;
        for $l = 1, ..., j-1$ do $\mu_{k,l} := \mu_{k,l} - r\mu_{j,l}$;
        end for;
        $\mu_{k,j} = \mu_{k,j} - r$;
      end for;
      $k := k + 1$;
    else *(Fall 1)*
      vertausche b_k mit b_{k-1} sowie T_k mit T_{k-1};
      $k := k - 1$;
    end if;
  else
  $k := k + 1$;
  end if;
end while;

Satz 3.10.
Der LLL-Algorithmus terminiert.

Beweis.
Sei $d_i := det((b_j, b_l))_{1 \leq j,l \leq i}$ für $0 \leq i \leq n$. Es gilt $d_i = \prod_{j=1}^{i} \left| b_j^* \right|^2$ für $0 \leq i \leq n$. Die d_i sind also positive reelle Zahlen. Es gilt $d_0 = 1$ und $d_n = d(L)^2$. Setze $D := \prod_{i=1}^{n-1} d_i$.
Die Zahl D verändert sich nur, falls sich die b_i^* ändern. Dies passiert nur im Fall 1, wobei die Zahl d_{k-1} um einen Faktor $< \frac{3}{4}$ reduziert wird. Die anderen d_i bleiben davon unberührt. Damit wird insgesamt D um einen Faktor $< \frac{3}{4}$ reduziert.
Es wird gezeigt, dass eine positive untere Schranke für die d_i existiert, welche nur von L abhängt. Daraus folgt, dass es eine positive untere Schranke für D gibt, sowie eine obere Schranke für die Anzahl der Fälle, in denen der Fall 1 eintritt.
Im Fall 1 wird der Laufindex k um 1 reduziert, wobei er im Fall 2 um 1 erhöht wird. Der Laufindex startet bei $k = 2$ und ist beschränkt durch $n + 1$, also gilt für den gesamten Algorithmus $k \leq n + 1$. Also wird Fall 2 höchstens $n - 1$-mal öfter eintreten, als der Fall 1 und damit die Anzahl seines Eintretens beschränkt. Daraus folgt, dass der Algorithmus terminiert.
Um zu zeigen, dass die d_i eine untere Schranke haben, definiere

$$m(L) := min \left\{ |x|^2 : \ x \in L, \ x \neq 0 \right\}.$$

Dies ist eine positive reelle Zahl. Für $i > 0$ kann man die d_i als das Quadrat der Determinante des Gitters vom Rang i, welches von $b_1, b_2, ..., b_i$ in $\sum_{j=1}^{i} \mathbb{R} b_j$ aufgespannt wird, interpretieren. Dieses Gitter enthält einen Vektor $x \neq 0$, mit $|x|^2 \leq (4/3)^{(i-1)/2} d_i^{1/i}$ [Cas71]. Damit gilt $d_i \geq (3/4)^{i(i-1)/2} m(L)^i$, welches die erforderliche untere Schranke des Algorithmus ist.

□

Damit ist bewiesen, dass jedes Gitter eine LLL-reduzierte Basis besitzt und dass diese mit Hilfe des LLL-Algorithmus berechnet werden kann.

Der LLL-Algorithmus läßt sich so umformulieren, dass er nur mit Gram-Matrizen arbeitet. Dafür müssen einige Stellen im ursprünglichen LLL-Algorithmus abgeändert werden, in dem Sinne, dass nicht mehr mit den Vektoren einer beliebigen Gitterbasis, sondern mit der Gram-Matrix der Basis gearbeitet wird.

4 Der Zusammenhang zwischen Faktoren und Gittern

In diesem Abschnitt definieren wir p als eine Primzahl und mit $g = \sum_i a_i X^i \in \mathbb{Z}[X]$ ist $(g \ mod \ p^k)$ das Polynom $\sum_i (a_i \ mod \ p^k) X^i \in (\mathbb{Z}/p^k\mathbb{Z})[X]$. Wir nehmen weiterhin ein Polynom $f \in \mathbb{Z}[X]$ von *Grad* n, $n > 0$ an und ein Polynom $h \in \mathbb{Z}[X]$, welches im gesamten Kapitel folgenden vier Eigenschaften genügen wird:

(A) h ist normiert,
(B) $(h \ mod \ p^k) \mid (f \ mod \ p^k$ in $(\mathbb{Z}/p^k\mathbb{Z})[X]$,
(C) $(h \ mod \ p)$ ist irreduzibel in $\mathbb{F}_p[X]$,
(D) $(h \ mod \ p)^2 \nmid (f \ mod \ p)$ in $\mathbb{F}_p[X]$.

Wir definieren $l := deg(h)$, d.h. $0 < l \leq n$.

Proposition 4.1.
Das Polynom f hat einen irreduziblen Faktor $h_0 \in \mathbb{Z}[X]$ mit $(h \ mod \ p) \mid (h_0 \ mod \ p)$. Dieser Teiler ist eindeutig bis auf Vorzeichen bestimmt.
Falls $g \mid f$ in $\mathbb{Z}[X]$, dann sind folgende drei Aussagen äquivalent:

$$(i) \ (h \ mod \ p) \mid (g \ mod \ p) \text{ in } \mathbb{F}_p[X],$$
$$(ii) \ (h \ mod \ p^k) \mid (g \ mod \ p^k) \text{ in } (\mathbb{Z}/p^k\mathbb{Z})[X],$$
$$(iii) \ h_0 \mid g \text{ in } \mathbb{Z}[X].$$

Insbesondere gilt $(h \ mod \ p^k) \mid (h_0 \ mod \ p^k)$ in $(\mathbb{Z}/p^k\mathbb{Z})[X]$.

Beweis.
Die Existenz eines solchen h_0 folgt aus den Forderungen (B) und (C) und die Eindeutigkeit aus (D).
Zu $(i) \to (ii)$ und $(i) \to (iii)$:
Mit (i) und (D) folgt, dass $(h \ mod \ p) \nmid (f/g \ mod \ p)$ in $\mathbb{F}_p[X]$. Also, da $h_0 \nmid f/g$ in $\mathbb{Z}[X]$ gilt, muss es g teilen. Dies ist die Behauptung von (iii).
Mit (C) sind $(h \ mod \ p)$ und $(f/g \ mod \ p)$ teilerfremd in $\mathbb{F}_p[X]$, wodurch wir auch $(\lambda_1 \ mod \ p) \cdot (h \ mod \ p) + (\mu_1 \ mod \ p) \cdot (f/g \ mod \ p) = 1$ für bestimmtes $\lambda_1, \mu_1 \in \mathbb{Z}$ erhalten. Daher gilt $\lambda_1 h + \mu_1 f/g = 1 - pv_1$ für bestimmte $v_1 \in \mathbb{Z}[X]$. Multiplizieren wir dies mit $1 + pv_1 + p^2 v_1^2 + ... + p^{k-1} v_1^{k-1}$ und g, so erhalten wir $\lambda_2 h + \mu_2 f \equiv g \ mod \ p^k\mathbb{Z}[X]$ für bestimmte $\lambda_2, \mu_2 \in \mathbb{Z}$. Da die linke Seite modulo p^k teilbar ist durch $(h \ mod \ p^k)$, gilt dies auch für die rechte Seite. Dies beweist (ii).
Die letzte Aussage folgt mit $g = h_0$. □

Im Folgenden sei $m \in \mathbb{Z}$ mit $m \geq l$ und sei $L := \{p \in \mathbb{Z}[X] : Grad(p) \leq m\}$ die Menge aller ganzzahligen Polynome mit $Grad \leq m$, für die gilt, das sie teilbar sind durch $h \ mod \ p^k$, falls sie $modulo \ p^k$ in $\mathbb{Z}/p^k\mathbb{Z}[X]$ genommen werden. Dies ist eine

Teilmenge des Vektorraums $\mathbb{R}^{m+1}$. L ist ein Gitter in $\mathbb{R}^{m+1}$ mit Basis:

$$\left\{p^k X^i : \ 0 \leq i < l\right\} \cup \left\{hX^j : \ 0 \leq j \leq m-l\right\}.$$

Aus der Definition eines Gitters folgt $d(L) = p^{kl}$.

Wir definieren für ein Polynom $\sum_i a_i X^i \in \mathbb{R}[X]$: $\left|\sum_i a_i X^i\right| := (\sum_i a_i^2)^{1/2}$.

Proposition 4.2.
Sei $b \in L$ mit $p^{kl} > |f|^m |b|^n$.
Dann gilt $h_0 \mid b$ in $\mathbb{Z}[X]$ und insbesondere gilt $ggT(f,b) \neq 1$.

Beweis.
Sei oBdA $b \neq 0$. Sei $g := ggT(f,b)$. Es ist zu zeigen, dass $(h \ mod \ p) \mid (g \ mod \ p)$ gilt.
Wir nehmen an, dass $(h \ mod \ p) \nmid (g \ mod \ p)$ gilt, beweisen also durch Widerspruch.
Dann gilt $\lambda_3 h + \mu_3 g = 1 - pv_3$ für bestimmte $\lambda_3, \mu_3, v_3 \in \mathbb{Z}[X]$. Sei $e = deg(g)$ und $m' = deg(b)$. Dann gilt $0 \leq e \leq m' \leq m$. Definiere:

$$\begin{aligned} M &:= \left\{\lambda f + \mu b : \ \lambda, \mu \in \mathbb{Z}[X], \ deg(\lambda) < m' - e, deg(\mu) < n - e\right\} \\ &\subset \mathbb{Z} + \mathbb{Z} \cdot X + ... + \mathbb{Z} \cdot X^{n+m'-e-1}. \end{aligned}$$

Sei M' die Projektion auf

$$\mathbb{Z} \cdot X^e + \mathbb{Z} \cdot X^{e+1} + ... + \mathbb{Z} \cdot X^{n+m'-e-1}.$$

Nehmen wir an, dass $\lambda f + \mu b$ auf Null abbildet in M', mit λ, μ wie in M. Dann folgt, dass $deg(\lambda f + \mu b) < e$, aber es gilt $g \mid (\lambda f + \mu b)$. Also ist $\lambda f + \mu b = 0$. Aus $\lambda \cdot (f/g) = -\mu \cdot (b/g)$ und $ggT(f/g, b/g) = 1$ folgt $f/g \mid \mu$. Aber es gilt $deg(\mu) < n - e = deg(f/g)$, also ist $\mu = 0$ und damit auch $\lambda = 0$.
Also sind die Projektionen von $\left\{X^i f : \ 0 \leq i < m' - e\right\} \cup \left\{X^j b : \ 0 \leq j < n - e\right\}$ auf M' linear unabhängig. Da diese Projektionen bereits M' aufspannen, folgt, dass M' ein Gitter vom Rang $n + m' - 2e$ ist.
Aus der Hadamard-Ungleichung und der Voraussetzung der Proposition folgt nun

$$d(M') \leq |f|^{m'-e} \cdot |b|^{n-e} \leq |f|^m |b|^n < p^{kl}.$$

Nun werden wir zeigen, dass aus $\lambda_3 h + \mu_3 g = 1 - pv_3$ (wie oben)

$$\{v \in M : \ deg(v) < e + l\} \subset p^k \mathbb{Z}[X]$$

folgt.
Sei $(b_e, b_{e+1}, ..., b_{n+m'-e-1})$ eine Basis von M' mit $deg(b_j) = j$. Dann sind die Leitkoeffizienten von $b_e, b_{e+1}, ..., b_{e+l-1}$ teilbar durch p^k. Da $d(M')$ gleich dem Produkt der

Leitkoeffizienten von $b_e, b_{e+1}, ..., b_{n+m'-e-1}$ ist, folgt $d(M') \geq p^{kl}$. Dies steht aber im Widerspruch zu $d(M') < p^{kl}$.
Sei nun $v \in M$, $deg(v) < e+l$. Dann gilt $g \mid v$. Durch Multiplizieren mit v/g und $1 + pv_3 + p^2v_3^2 + ... + p^{k-1}v_3^{k-1}$ erhalten wir

$$\lambda_4 h + \mu_4 v \equiv v/g \ mod \ p^k\mathbb{Z}[X]$$

mit $\lambda_4, \mu_4 \in \mathbb{Z}[X]$. Aus $v \in M$ und $b \in L$ folgt, dass $(h \ mod \ p^k) \mid (v \ mod \ p^k)$ gilt. Also folgt jetzt auch $(h \ mod \ p^k) \mid (v/g \ mod \ p^k)$. Aber $(h \ mod \ p^k)$ ist normiert und hat Grad l, wobei $(v/g \ mod \ p^k)$ Grad $< e+l-e = l$ hat. Also gilt $v/g \equiv 0 \ mod \ p^k\mathbb{Z}[X]$ und damit $v \equiv 0 \ mod \ p^k\mathbb{Z}[X]$. □

Proposition 4.3.
Sei $b_1, b_2, ..., b_{m+1}$ eine LLL-reduzierte Basis von L und

$$p^{kl} > 2^{mn/2}\binom{2m}{m}^{n/2} |f|^{m+n}.$$

Dann gilt $deg(h_0) \leq m \ \Leftrightarrow |b_1| < (p^{kl}/\,|f|^m)^{1/n}$ für h_0 wie in (4.1).

Beweis.
für "⇒": Sei $deg(h_0) \leq m$. Dann ist $h_0 \in L$ und es gilt $|h_0| \leq \binom{2m}{m}^{1/2} \cdot |f|$ aufgrund von Mignotte [Mig74]. Es folgt aus dem ersten Teil (LLL-Algorithmus), dass $|b_1| \leq 2^{m/2} \cdot |h_0| \leq 2^{m/2} \cdot \binom{2m}{m}^{1/2} \cdot |f|$. Mit der Voraussetzung folgt die Behauptung.
für "⇐": Folgt aus Proposition (4.2), da $deg(b_1) \leq m$. □

Proposition 4.4.
Existiere zusätzlich zu den Voraussetzungen der vorherigen Proposition (4.3) ein Index $j \in \{1, 2, ..., m+1\}$, für den $|b_j| < (p^{kl}/\,|f|^m)^{1/n}$ gilt. Sei t das größte dieser j. Dann gilt:

$$deg(h_0) = m + 1 - t$$
$$h_0 = ggT(b_1, b_2, ..., b_n)$$

und $|b_j| < (p^{kl}/\,|f|^m)^{1/n}$ für alle $1 \leq j \leq t$.

Beweis.
Sei $J = \{j \in \{1, 2, ..., m+1\} : \ |b_j| < (p^{kl}/\,|f|^m)^{1/n}\}$. Es gilt $h_0 \mid b_j, \ \forall j \in J$. Sei $h_1 := ggT(\{b_j : \ j \in J\})$, dann gilt $h_0 \mid h_1$. h_1 teilt jedes b_j, $j \in J$ und hat Grad $\leq m$, gehört also zu

$$\mathbb{Z} \cdot h_1 + \mathbb{Z} \cdot h_1X + ... + \mathbb{Z} \cdot h_1X^{m-deg(h_1)}.$$

Die b_j sind linear unabhängig, daher folgt $\#J \leq m+1-deg(h_1)$.
Mit Mignotte [Mig74] aus dem letzten Beweis folgt $\left|h_0 X^i\right| = |h_0| \leq \binom{2m}{m}^{1/2} \cdot |f|$ für alle $i \geq 0$. Für $i = 0, 1, ..., m - deg(h_0)$ gilt $h_0 X^i \in L$ und es folgt

$$\left|b_j\right| \leq 2^{m/2} \cdot \binom{2m}{m}^{1/2} \cdot |f|$$

für $1 \leq j \leq m+1-deg(h_0)$. Mit $p^{kl} > 2^{mn/2}\binom{2m}{m}^{n/2} |f|^{m+n}$ folgt dann:

$$\{1, 2, ..., m+1-deg(h_0)\} \subset J.$$

Es folgt nun mit $h_0 \mid h_1$, dass $deg(h_0) = deg(h_1) = m+1-t,\ J = \{1, 2, ..., t\}$.
Es ist noch zu zeigen, dass h_0 und h_1 bis auf Vorzeichen gleich sind. Dazu zeigen wir, dass h_1 primitiv ist. Wähle $j \in J$ und sei d_j der Inhalt von b_j. Dann gilt $h_0 \mid b_j/d_j$ und da $h_0 \in L$ folgt $b_j/d_j \in L$. Da aber b_j aus einer Basis von L ist, gilt $d_j = 1$ und b_j ist primitiv. Das selbe gilt für den Teiler h_1 von b_j. □

Wir möchten den irreduziblen Faktor h_0 des Polynoms f bestimmen. Dies bedeutet, dass wir das h_0 aus der Proposition (4.1) bestimmen wollen. Die Bedingung
$(h \ mod \ p^k) \mid (h_0 \ mod \ p^k)$ bedeutet, dass

$$h_0 \ mod \ p^k = (h \ mod \ p^k)(g \ mod \ p^k) = g_0 h + g_1 X h + ... + g_j X^j h \ mod \ p^k$$

mit $g_0, ..., g_j \in \mathbb{Z}$ wobei $j := m - l,\ m := Grad(h_0),\ l := Grad(h)$. Also liegt h_0 in dem von

$$\left\{p^k X^i :\ 0 \leq i < l\right\} \cup \left\{h X^j :\ 0 \leq j \leq m-l\right\}$$
$$= (h, Xh, ..., X^j h, p^k, p^k X, ..., p^k X^j)$$

aufgespannten Gitter L der Dimension $m+1$.
Wir identifizieren nun die Polynome in $\mathbb{Z}[X]$ vom Grad $\leq m$ mit ihren Koeffizienten (z.B.: $h =: \sum_{i=0}^{l} c_i X^i$ mit $(c_0, ..., c_l, 0, ..., 0)$ und $h_0 =: \sum_{i=0}^{l} v_i X^i$ mit $(v_0, ..., V_m)$). Dann wird das Gitter L durch

$$(c_0, ..., c_l, 0, ..., 0), (o, c_0, ..., c_l, 0, ..., 0), ..., (0, ..., 0, c_0, ..., c_l), (p^k, 0, ..., 0), ..., (0, ..., 0, p^k, 0, ..., 0)$$

erzeugt. Sind die Koeffizienten von h_0 im Vergleich zu p^k klein, so ist h_0 ein kurzer Vektor in L. Also wenden wir auf obiges Gitter den LLL-Algorithmus an, mit dem wir eine reduzierte Basis $(b_1, ..., b_n)$ von L finden. h_0 lässt sich dann als $h_0 = ggT(b_1, ..., b_n)$ berechnen.

5 Faktorisieren von rationalen Polynomen

In diesem Abschnitt wird der endgültige Algorithmus beschrieben, mit dessen Hilfe man ein Polynom faktorisieren kann. Wir brauchen dafür jedoch einige Hilfsalgorithmen und Sätze, die innerhalb des Hauptalgorithmus zur Polynomfaktorisierung verwendet werden.

Definition 5.1.
Seien $f = f_m X^m + f_{m-1} X^{m-1} + ... + f_0$ und $g = g_n X^n + g_{n-1} X^{n-1} + ... + g_0$ zwei Polynome von Grad m bzw n aus $R[X]$, R kommutativer Ring mit 1. Dann ist die *Resultante* der beiden Polynome f, g definiert als

$$Res(f,g) = det \begin{pmatrix} f_m & f_{m-1} & \cdots & f_0 & & \\ & f_m & f_{m-1} & \cdots & f_0 & \\ & & & \ddots & & \\ & & f_m & f_{m-1} & \cdots & f_0 \\ g_n & g_{n-1} & \cdots & g_0 & & \\ & g_n & g_{n-1} & \cdots & g_0 & \\ & & & \ddots & & \\ & & g_n & g_{n-1} & \cdots & g_0 \end{pmatrix}$$

Es gilt: falls $Res(f,g) = 0$, so besitzen f und g einen gemeinsamen Faktor positiven Grades.

Satz 5.2. (Cantor, Zassenhaus für (p>2))
Eingabe: $A \in \mathbb{F}_p[X]$ Produkt von mindestens zwei verschiedenen irreduziblen Polynomen vom Grad d.
Ausgabe: Ein nicht-trivialer Faktor B von $A = B\frac{A}{B}$.
Algorithmus: $k := Grad(A)$;
while () do
  Wähle zufälliges $T \in \mathbb{F}_p[X]$ mit $Grad(T) \leq 2d - 1$;
  Berechne $B := ggT(T^{(p^d-1)/2}), A$ mit dem Euklidischen Algorithmus;
  if $k > Grad(B) > 0$ then $Return(B)$; end if;
end while;

Satz 5.3. (Berlekamp-Algorithmus)
Sei $A \in \mathbb{F}_p[X]$ quadratfrei, $n := Grad(A)$.
Bestimme eine $\mathbb{F}_p[X]$-Basis von $M := \left\{T \in \mathbb{F}_p[X] \mid Grad(T) < n,\ T^p \equiv T\ (mod\ A)\right\}$:
Ist $T(X) = \sum_{j=0}^{n-1} t_j X^j \in M$, so ist $T^p(X) = T(X^p) = \sum_{j=0}^{n-1} t_j X^{pj}$.
Berechne also die Matrix $Q = (q_{ij}) \in \mathbb{F}_p^{nxn}$ mit

$$X^{pj} \equiv \sum_{i=0}^{n-1} q_{ij} X^i\ (mod\ A)$$

Dann gilt $T^p \equiv T \ (mod \ A) \Leftrightarrow (t_0, t_1, ..., t_{n-1}) \in Kern(Q - I_n)$.
Bestimme also eine $\mathbb{F}_p[X]$-Basis von $Kern(Q - I_n)$ mit dem *Gauß-Algorithmus*. Da $1^p = 1$ gilt $\mathbb{F}_p \subset M$ (d.h. $(1, 0, ..., 0) \in Kern(Q - I_n)$). Jeder von 1 linear unabhängige Vektor $t \in Kern(Q - I_n)$ entspricht aber einem nicht konstanten $T \in M$, für das Konstanten $s_i \in \mathbb{F}_p$ existieren mit $T \equiv s_i \ (modA_i)$. Die $s_i \in \mathbb{F}_p$ sind nicht alle gleich und

$$A = \prod_{s \in \mathbb{F}_p} ggT(T - s, A)$$

ist eine nicht-triviale Faktorisierung von A.

Für vollständige bewertete Körper, diskrete Bewertungen und Bewertungsringe siehe [Neb10].

Satz 5.4. (Henselsches Lemma)
Sei $(K, |.|)$ ein vollständiger nicht archimedisch bewerteter Körper mit Bewertungsring R und maximalen Ideal $\pi \trianglelefteq R$ und Restklassenkörper $F = R/\pi$. Sei $f(x) \in R[x]$ normiert und $^- : R[x] \to F[x]$ der natürliche Epimorphismus. Falls $\overline{f}(x) = h_0(x)g_0(x)$ mit $ggT(h_0, g_0) = 1$ so gibt es $h(x), g(x) \in R[x]$ mit $\overline{h} = h_0$, $\overline{g} = g_0$ und $f = gh$.

Beweis.
siehe [Neb10](S.69-70). □

Nun folgt der Algorithmus, mit dem wir ein primitives Polynom $f \in \mathbb{Z}[X]$ mit $Grad\ f > 0$ in irreduzible Faktoren in $\mathbb{Z}[X]$ zerlegen:

Algorithmus 5.5.
Eingabe: Ein primitives Polynom $f \in \mathbb{Z}[X]$ mit $Grad > 0$.
Ausgabe: eine Zerlegung des Polynoms f in irreduzible Faktoren in $\mathbb{Z}[X]$
Algorithmus: Bestimme $Res(f, f')$;
if $Res(f, f') = 0$ then
  definiere $g := ggT(f, f')$ in $\mathbb{Z}[X]$; $f_0 := f/g$ (multiple Faktoren werden herausgenommen)
else
  setze $f_0 := f$
end if;
Bestimme $p \in \mathbb{P}$ mit $p \nmid Res(f_0, f_0')$;
Zerlege $(f_0\ mod\ p)$ in irreduzible Faktoren $h_1, ..., h_{n'}$ in $\mathbb{F}_p[X]$ mit Hilfe des Berlekamp-Algorithmus (5.3), definiere $f_1 := 1$; $f_2 := f_0$;
while $\{h_1, ..., h_{n'}\} \neq \varnothing$ do
  wähle einen irreduziblen Faktor h aus $\{h_1, ...h_{n'}\}$
  setze $l := deg(h)$;
  if $(l < n)$ then
    sei $k \in \mathbb{Z}$ minimal mit $p^{kl} > 2^{(n-1)n/2} \cdot \binom{2(n-1)}{n-1}^{n/2} \cdot |f|^{2n-1}$
    lifte $(h\ mod\ p)$ zu $(h\ mod\ p^k)$ mit dem Henselschen Lemma (5.4)
    definiere $u \in \mathbb{Z}$ maximal mit $l \leq (n-1)/2^u$
    $h_0 := 1$ (für den Fall, dass f irreduzibel ist)
    for $j = u, ..., 0$
      definiere $m := (n-1)/2^j$; $L := (h, Xh, ..., X^j h, p^k, p^k X, ..., p^k X^j)$;
      Sei $(b_1, ..., b_{m+1})$ eine LLL-reduzierte Basis von L
      if $|b_1| < (p^{kl}/|f|^m)^{1/n}$ then
        $h_0 := ggT(b_1, ..., b_t)$ mit $Grad(h_0) = m + 1 - t$;
        break;
      end if;
    end for;
  else
    $h_0 = f_2$;
    break;
  end if;
  setze $f_1 := f_1 h_0$; $f_2 := f_2/h_0$;
  streiche h aus $\{h_1, ...h_{n'}\}$
end while;
return(f_1);

Da die irreduziblen Faktoren nun bekannt sind, können wir das Polynom g, mit $f_0 g = f$, noch faktorisieren und wir haben die komplette Faktorisierung des Polynoms $f \in \mathbb{Z}[X]$ mit *Grad* $f > 0$ in irreduzible Faktoren in $\mathbb{Z}[X]$ erhalten.

6 Kurzüberblick über die Entwicklung von Faktorisierungsalgorithmen seit 1969

Wenn wir die Laufzeit und Effizienz eines Algorithmus betrachten, so müssen wir die Eingangswerte bewerten und für den speziellen Algorithmus, die Kosten der einzelnen Operationen und die Anzahl solcher Operationen berechnen. Dies ist in unserem Fall zum Beispiel der Grad des Polynoms sowie die Größe der einzelnen Koeffizienten. Bis der *LLL-Algorithmus* im Jahr 1982 von Arjen Lenstra, Hendrik Lenstra und Laszlo Lovasz entwickelt wurde, war der 1969 entwickelte Algorithmus von Hans Zassenhaus, der sogenannte *Zassenhaus-Algorithmus*, der effizienteste für ein Computeralgebrasystem bekannte Algorithmus. Die Veröffentlichung des LLL-Algorithmus gab dem Computer ein neues Werkzeug, um Polynome zu faktorisieren. Der theoretische *worst case*, also die Beschränkung der Laufzeit des Algortihmus, bestimmt aus der Komplexität und Größe der Eingangswerte und die damit verbundene obere Komplexitätsgrenze waren besser als die im Zassenhaus-Algorithmus errechnete exponentielle Komplexitätsgrenze. Für den LLL-Algorithmus konnte bewiesen werden, dass er eine polynomielle obere Komplexitätsgrenze besitzt. Der Zassenhaus-Algorithmus kann jedoch ein Polynom in der Praxis, das bedeutet im durchnittlichen Fall, schneller in seine irreduziblen Faktoren zerlegen als der später entwickelte LLL-Algorithmus. Daher wurde in Computeralgebrasystemen der in der Theorie schlechtere Algorithmus implementiert.

2002 löste der *van Hoeij-Algorithmus* den Zassenhaus-Algorithmus bezüglich der durschnittlich schnellsten Laufzeit ab und wurde somit in der Praxis zum schnellsten Algorithmus für das Problem der Polynomfaktorisierung. Der van Hoeij-Algorithmus ist eine Variation des Zassenhaus-Algorithmus. Im Zassenhaus-Algorithmus war der teuerste Teil bezüglich der Laufzeit, bestimmte Vektoren $\{0,1\}^n$ zu errechnen, welche die Kombination der irreduziblen Faktoren des Polynoms in $\mathbb{F}_p[X]$ beschreiben. Mark van Hoeij kombinierte den Zassenhaus-Algorithmus mit dem LLL-Algorithmus, in dem er die gesuchten Vektoren aus speziellen Gittern, den sogennanten *knapsack lattices*, mittels der LLL-Reduktion errechnete. Van Hoeij konnte jedoch keine obere Komplexitätsgrenze für seinen Algorithmus angeben, welche aus dem van Hoeij-Algorithmus auch in der Theorie einen besseren Algorithmus als den LLL-Algorithmus hätte machen können. Er konnte nur beweisen, dass sein Algorithmus terminiert.

2004 wurde der Algorithmus in der Praxis von dem *Belabas-Algorithmus* abgelöst. In einer von Karim Belabas, Mark von Hoeij, Jürgen Klüners und Allan Steel überarbeiteten Version des van Hoeij-Algorithmus konnte eine polynomielle Komplexitätsgrenze bewiesen werden. Es wurde ein neuer Begriff einer reduzierten Basis, der sogenannten B-reduzierten Basis eingeführt. Jedoch arbeitete dieser Algorithmus mit sehr großen, umständlich zu errechnenden Gittern, wodurch in der Praxis eine Verbesserung erzielt werden konnte, jedoch der LLL-Algorithmus theoretisch

schneller war. Sie fanden heraus, dass mit kleinen Zahlen und vielen LLL-Reduktionsaufrufen die besten Laufzeiten in der Praxis erreicht werden können.

2008 wurde der *van Hoeij/Novocin-Algorithmus* entwickelt. Dies ist eine Weiterentwicklung des Belabas-Algorithmus, welcher wiederum auf dem ursprünglichen van Hoeij-Algorithmus basierte. Da der Belabas-Algorithmus auf der Basis von vielen LLL-Reduktionsaufrufen mit kleinen Zahlen basiert, wählt der neue Algorithmus während dieser Schritte die brauchbaren aus und sichert dadurch, dass kein LLL-Reduktionsaufruf zuviel gemacht wird und jeder Aufruf verwendet werden kann. Der neue Algorithmus läuft in polynomieller Zeit und hat den LLL-Algorithmus in der Theorie und den Belabas-Algorithmus in der Praxis abeglöst. Somit ist zum ersten Mal ein Algorithmus für die Polynomfaktorisierung sowohl in der Praxis als auch in der Theorie am schnellsten.

LLL-Reduktion in Maple-Code

```
LLL := proc(B::Matrix)
local n,Bred,Borth,bk,T,tk,L,k,mu,r,a,b,j,p,l;
n:=ColumnDimension(B);
T:=IdentityMatrix(n); #Transformationsmatrix
Bred:=B; #Kopiekonstruktor des unreduzierten Gitters, wird zur reduzierten Basis
Borth:=B; #Othogonale Matrix (ueber Gram-Schmidt) von B
k:=2; #Laufindex
mu:=<IdentityMatrix(n)>; #Matrix(n,n,[]); #mu_i,j
while k<=n do
  Print(k):
  if k>=2 then
    Print(mu[k,k-1]):
    mu[k,k-1]:=(Column(Bred,k).Column(Borth,k-1))/abs(Column(Borth,k-1).\
      Column(Borth,k-1)); #Berechnung der Gram-Schmidt-Faktoren
    if abs(mu[k,k-1])>(1/2) then
      r:=round(mu[k,k-1]);
      Bred:=<SubMatrix(Bred,[1..n],[1..k-1])|Column(Bred,k)-r*Column(Bred,k-1)|\
        SubMatrix(Bred,[1..n],[k+1..n])>;
      T:=<SubMatrix(T,[1..n],[1..k-1])|Column(T,k)-r*Column(T,k-1)|\
        SubMatrix(T,[1..n],[k+1..n])>;
      for j from 1 to k-2 do
        mu[k,j]:=mu[k,j]-r*mu[k-1,j];
      od;
      mu[k,k-1]:=mu[k,k-1]-r;
    fi;
    for j from 1 to (k-2) do #Berechnung der GramSchmidt-Faktoren
      mu[k,j]:=evalf((Column(Bred,k).Column(Borth,j))/abs(Column(Borth,j).\
        Column(Borth,j)));
    od;
    Borth:=<SubMatrix(Borth,[1..n],[1..(k-1)])|Column(Bred,k) -\
      add((mu[k,o])*Column(Borth,o),o=1..k-1)|SubMatrix(Borth,[1..n],[(k+1)..n])>;
    if (Column(Borth,k).Column(Borth,k)+(mu[k,k-1])^2*Column(Borth,k-1).\
      Column(Borth,k-1)) >= (3/4 * Column(Borth,k-1).Column(Borth,k-1)) then
      for p from (k-2) by (-1) to 1 do
        r:=round(mu[k,p]);
        Bred:=<SubMatrix(Bred,[1..n],[1..k-1])|Column(Bred,k)-r*Column(Bred,p)|\
          SubMatrix(Bred,[1..n],[k+1..n])>;
        T:=<SubMatrix(T,[1..n],[1..k-1])|Column(T,k)-r*Column(T,p)|\
          SubMatrix(T,[1..n],[k+1..n])>;
        for j from 1 to p-1 do
          mu[k,j]:=mu[k,j] - r*mu[p,j]
        od;
        mu[k,p]:=mu[k,p] - r;
      od;
      k:=k+1;
    else
      Bred:=<SubMatrix(Bred,[1..n],[1..k-2])|Column(Bred,k)|Column(Bred,k-1)|\
        SubMatrix(Bred,[1..n],[k+1..n])>;
      T:=<SubMatrix(T,[1..n],[1..k-2])|Column(T,k)|Column(T,k-1)|\
        SubMatrix(T,[1..n],[k+1..n])>;
      k:=k-1;
    fi;
  else
    k:=k+1;
  fi;
od;
return(Bred,T,mu,Borth);
end:
```

Anwendung des Algorithmus LLL auf ein Beispiel zur Reduktion einer Basis eines Gitters in seine LLL-reduzierte Form nach Henri Cohen (A course in computational algebraic number theory. GTM. 138. Springer, 2000). L[1] ist die LLL-reduzierte Gitterbasis, L[2] ist die Transformationsmatrix mit T.L[2]=L[1] und L[4] ist eine orthogonale Basis von T. In L[3] sind die mu_ij des Gram-Schmidt-Othogonalisierungsverfahren angegeben.

```
> T:=Matrix(3,3,[1,-1,3,1,0,5,1,2,6]);
> L:=LLL(T);
```

$$T := \begin{bmatrix} 1 & -1 & 3 \\ 1 & 0 & 5 \\ 1 & 2 & 6 \end{bmatrix}$$

$$L := \begin{bmatrix} 0 & 1 & -1 \\ 1 & -1 & 0 \\ 0 & 1 & 2 \end{bmatrix}, \begin{bmatrix} -4 & 9 & 0 \\ -1 & 2 & 1 \\ 1 & -2 & 0 \end{bmatrix}, \begin{bmatrix} 1 & 0 & 0 \\ \frac{-1}{3} & 1 & 0 \\ 0.3333333333 & \frac{1}{3} & 1 \end{bmatrix}, \begin{bmatrix} 1 & \frac{4}{3} & -1.77777777774444435 \\ 1 & \frac{-2}{3} & -0.111111111077777769 \\ 1 & \frac{4}{3} & 1.22222222225555566 \end{bmatrix}$$

```
> T.L[2]=L[1];
```

$$\begin{bmatrix} 0 & 1 & -1 \\ 1 & -1 & 0 \\ 0 & 1 & 2 \end{bmatrix} = \begin{bmatrix} 0 & 1 & -1 \\ 1 & -1 & 0 \\ 0 & 1 & 2 \end{bmatrix}$$

Der LLL-Algorithmus wird zum Faktorisieren von Polynomen verwendet. Im Folgenden wird das Polynom **f=x^4+x^3-x-1** mithilfe des LLL-Algorithmus **faktorisiert**. Die Bezeichnung ist analog zur Ausarbeitung des Vortrags und im Speziellen des Algorithmus (5.6). n bezeichnet den Grad von f.

```
> f:=expand((x^2-1)*(x^2+x+1));
> n:=degree(f);
> factor(f);
```

$$f := x^4 + x^3 - x - 1$$

$$n := 4$$

$$(x-1)\,(x+1)\,(x^2+x+1)$$

Der Betrag des Polynoms, also die Euklidische Norm der Koeffizienten ist:

```
> absf:=sqrt(1^2+1^2+1^2+1^2);
```

$$absf := 2$$

f ist quadratfrei, da:

```
> gcd(f,diff(f,x));
```

$$1$$

Gesucht ist die kleinste Primzahl p, welche die Resultante Res(f,f')=-108=-2^2*3^3 nicht teilt:

```
> resultant(f,diff(f,x),x);
> p:=5;
```

$$-108$$

$$p := 5$$

Das Polynom f wird nun mit Hilfe des Berlekamp-Algorithmus ueber F_5[X] in irreduzible Faktoren zerlegt.

```
> Berlekamp(f,x) mod p;
```

$$\{x^2 + x + 1,\ x + 1,\ x + 4\}$$

Betrachten wir nun den irreduziblen Faktor **h:=x^2+x+1** mod 5 mit seinem Grad(h)=2=:l

```
> h:=x^2+x+1;
> l:=degree(h);
```

$$h := x^2 + x + 1$$

$$l := 2$$

Wie suchen nun ein k aus Z welches

```
> p^(k*l)>2^((n-1)*n/2)*binomial(2*(n-1),(n-1))^(n/2)*absf^(2*n-1):
```

erfuellt. Dieses k ergibt sich als 5, da

```
> k:=4;
> evalb(evalf(p^(k*l))>evalf(2^((n-1)*n/2)*binomial(2*(n-1),(n-1))^(n/2)
> *absf^(2*n-1)));
> k:=5;
> evalb(evalf(p^(k*l))>evalf(2^((n-1)*n/2)*binomial(2*(n-1),(n-1))^(n/2)
> *absf^(2*n-1)));
```

$$k := 4$$

$$false$$

$$k := 5$$

$$true$$

Wir liften nun den Faktor h mod 5 mit Hilfe des Henselschen Lemmas des Programms Magma zu einem Faktor h mod 5^5:

```
> h:=x^2+x+1;
```

$$h := x^2 + x + 1$$

Es wird ein maximales u gesucht, welches

```
> for u from 0 to 1 do
> print(u);
> evalb(l<=(n-1)/2^u);
> end do;
```

$$0$$

$$true$$

$$1$$

$$false$$

erfuellt. Dieses u ist also

```
> u:=0;
```

$$u := 0$$

Bestimme nun das Gitter L, welches h enthaellt, indem wir die Koeffizienten des Polynoms mit den Eintraegen der Basiselemente identifizieren und die m, fuer die Anzahl der Elemente der LLL-reduzierten Basis von L

```
> for j from u by (-1) to 0 do
> m:=floor((n-1)/2^j);
> end do;
```

$$m := 3$$

Mit m=3 bestimmen wir aus L

```
> T:=Transpose(Matrix(4,4,[1,1,1,0,0,1,1,1,5^5,0,0,0,0,5^5,0,0]));
```

$$T := \begin{bmatrix} 1 & 0 & 3125 & 0 \\ 1 & 1 & 0 & 3125 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

mit Hilfe der LLL-Reduktion eine LLL-reduzierte Basis:

```
> SubMatrix(LLL(T)[1],[1..4],[1..2]);
```

$$\begin{bmatrix} -1 & 2 \\ 0 & 1 \\ 0 & 1 \\ 1 & -1 \end{bmatrix}$$

Die ersten zwei Vektoren der LLL-reduzierten Basis erfuellen

```
> if evalf(sqrt(2^2+1^2+1^2+1^2))<evalf((p^(k*l)/absf^m)^(1/n))
then
> h_0:=gcd(x^3-1,-x^3+x^2+x+2);
> end if;
```

$$h\_0 := x^2 + x + 1$$

so dass wir einen irreduziblen Faktor von f gefunden haben.

Betrachten wir nun den irreduziblen Faktor **h:=x+1** mod 5 mit seinem Grad(h)=1=:l

```
> h:=x+1;
> l:=degree(h);
```

$$h := x + 1$$
$$l := 1$$

Wie suchen nun ein k aus Z welches

```
> p^(k*l)>2^((n-1)*n/2)*binomial(2*(n-1),(n-1))^(n/2)*absf^(2*n-1):
```

erfuellt. Dieses k ergibt sich als 10, da

```
> k:=9;
> evalb(evalf(p^(k*l))>evalf(2^((n-1)*n/2)*binomial(2*(n-1),(n-1))^(n/2)
> *absf^(2*n-1)));
> k:=10;
> evalb(evalf(p^(k*l))>evalf(2^((n-1)*n/2)*binomial(2*(n-1),(n-1))^(n/2)
> *absf^(2*n-1)));
```

$$k := 9$$
$$\mathit{false}$$
$$k := 10$$
$$\mathit{true}$$

Wir liften nun den Faktor h mod 5 mit Hilfe des Henselschen Lemmas des Programms Magma zu einem Faktor h mod 5^10:

```
> h:=x+1;
```

$$h := x + 1$$

Es wird ein maximales u gesucht, welches

```
> for u from 1 to 2 do
> print(u);
> evalb(l<=(n-1)/2^u);
> end do;
```

$$1$$
$$\mathit{true}$$
$$2$$
$$\mathit{false}$$

erfuellt. Dieses u ist also

```
> u:=1;
```

$$u := 1$$

Bestimme nun das Gitter L, welches h enthaellt, indem wir die Koeffizienten des Polynoms mit den Eintraegen der Basiselemente identifizieren und die m, mit der wir die Anzahl der Elemente der LLL-reduzierten Basis von L bestimmen koennen

```
> for j from u by (-1) to 0 do
> m:=floor((n-1)/2^j);
> end do;
```

$$m := 1$$
$$m := 3$$

Beginnend mit m=3 bestimmen wir aus L

```
> T:=Transpose(Matrix(4,4,[1,1,0,0,0,1,1,0,0,0,1,1,5^10,0,0,0]));
```

$$T := \begin{bmatrix} 1 & 0 & 0 & 9765625 \\ 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

mit Hilfe der LLL-Reduktion eine LLL-reduzierte Basis:

```
> SubMatrix(LLL(T)[1],[1..4],[1..3]);
```

$$\begin{bmatrix} 1 & 0 & 1 \\ 1 & 1 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Die ersten drei der Vektoren der LLL-reduzierten Basis erfuellen

```
> if evalf(sqrt(1^2+1^2+1^2))<evalf((p^(k*l)/absf^m)^(1/n)) then
> h_1:=gcd(x+1,gcd(x^2+x,x^3+1));
> end if;
```

$$h\_1 := x + 1$$

also erhalten wir einen irreduziblen Faktor h_1. (Haelt auch fuer m=1)

Betrachten wir nun den irreduziblen Faktor **h:=x+4** mod 5 mit seinem Grad(h)=1=:l

```
> h:=x+4;
> l:=degree(h);
```

$$h := x + 4$$
$$l := 1$$

Wie suchen nun ein k aus Z welches

```
> p^(k*l)>2^((n-1)*n/2)*binomial(2*(n-1),(n-1))^(n/2)*absf^(2*n-1):
```

erfuellt. Dieses k ergibt sich als 10, da

```
> k:=9;
> evalb(evalf(p^(k*l))>evalf(2^((n-1)*n/2)*binomial(2*(n-1),(n-1))^(n/2)
> *absf^(2*n-1)));
> k:=10;
> evalb(evalf(p^(k*l))>evalf(2^((n-1)*n/2)*binomial(2*(n-1),(n-1))^(n/2)
> *absf^(2*n-1)));
```

$$k := 9$$
$$\mathit{false}$$
$$k := 10$$
$$\mathit{true}$$

Wir liften nun den Faktor h mod 5 mit Hilfe des Henselschen Lemmas des Programms Magma zu einem Faktor h mod 5^11:

```
> h:=x+9765624;
```

$$h := x + 9765624$$

Es wird ein maximales u gesucht, welches

```
> for u from 1 to 2 do
> print(u);
> evalb(l<=(n-1)/2^u);
> end do;
```

$$1$$

$$\mathit{true}$$

$$2$$

$$\mathit{false}$$

erfullt. Dieses u ist also

```
> u:=1;
```

$$u := 1$$

Bestimme nun das Gitter L, welches h enthaellt, indem wir die Koeffizienten des Polynoms mit den Eintraegen der Basiselemente identifizieren und die m, fuer die Anzahl der Elemente der LLL-reduzierten Basis von L

```
> for j from u by (-1) to 0 do
> m:=floor((n-1)/2^j);
> end do;
```

$$m := 1$$

$$m := 3$$

Beginnend mit m=1 bestimmen wir aus L

```
> T:=Transpose(Matrix(4,4,[9765624,1,0,0,0,9765624,1,0,0,0,9765624,1,5^
> 10,0,0,0]));
```

$$T := \begin{bmatrix} 9765624 & 0 & 0 & 9765625 \\ 1 & 9765624 & 0 & 0 \\ 0 & 1 & 9765624 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

mit Hilfe der LLL-Reduktion eine LLL-reduzierte Basis:

```
> SubMatrix(LLL(T)[1],[1..4],[1..1]);
```

$$\begin{bmatrix} 1 \\ -1 \\ 0 \\ 0 \end{bmatrix}$$

Der erste der Vektoren der LLL-reduzierten Basis erfuellen

```
> if evalf(sqrt(1^2+1^2))<evalf((p^(k*l)/absf^m)^(1/n)) then
> h_2:=gcd(-x+1);
> end if;
```

$$h\_2 := x - 1$$

sodass wir einen irreduziblen Faktor h_2 erhalten.

Das Polynom faktorisiert als in:

```
> h_0*h_1*h_2=factor(f);
```

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

Literatur

[Cas71] J. W. S. Cassels. An introduction to the geometry of numbers. *Springer*, 1971.

[Klu] Jurgen Kluners. `http://www2.math.uni-paderborn.de/fileadmin/Mathematik/AG-Klueners/publications/factor_lll.pdf`.

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovasz. Factoring Polynomials with Rational Coefficients. *Springer, Mathematische Annalen 261*, pages 515–534, 1982.

[Mig74] M. Mignotte. An inequality about factors of ploynomials. *Math. Comp. 28*, pages 1153–1157, 1974.

[Neb00] Gabriele Nebe. Elementare Algorithmen in der Algebra (Vorlesung). *RWTH Aachen*, 2000.

[Neb10] Gabriele Nebe. Computeralgebra (Vorlesung). *RWTH Aachen*, pages 61–70, 2010.

[Nov] Andy Novocin. `http://andy.novocin.com/pro/complexityres.pdf` (von Hoeij/Novocin Algorithmus).

[NS] G. Nebe and N. Sloane. `http://www.math.rwth-aachen.de/~Gabriele.Nebe/LATTICES/` (Verzeichnis der benutzten Gitter).