Jahrbuch der Heinrich-Heine-Universität Düsseldorf

2007/2008

HEINRICH HEINE
UNIVERSITÄT
DÜSSELDORF

d|u|p

düsseldorf university press

Jahrbuch der Heinrich-Heine-Universität Düsseldorf

2007/2008

d|u|p

Jahrbuch der Heinrich-Heine-Universität Düsseldorf 2007/2008

Herausgegeben vom Rektor
der Heinrich-Heine-Universität Düsseldorf
Univ.-Prof. Dr. Dr. Alfons Labisch

Konzeption und Redaktion:
Univ.-Prof. em. Dr. Hans Süssmuth

d|u|p

Einbandgestaltung: Wiedemeier & Martin, Düsseldorf
Titelbild: Schloss Mickeln, Tagungszentrum der Universität
Redaktionsassistenz: Georg Stüttgen
Beratung: Friedrich-K. Unterweg
Satz: Friedhelm Sowa, LaTeX
Herstellung: Uniprint International BV, Meppel, Niederlande
Gesetzt aus der Adobe Times
ISBN 978-3-940671-10-3

Inhalt

Philosophische Fakultät

Wirtschaftswissenschaftliche Fakultät

Juristische Fakultät

Zentrale wissenschaftliche Einrichtungen der Heinrich-Heine-Universität Düsseldorf

Humanwissenschaftlich-Medizinisches Forschungszentrum
Zur Diskussion gestellt: Stammzellforschung

Gesellschaft von Freunden und Förderern der Heinrich-Heine-Universität Düsseldorf e.V.

Private Stiftungen für die Heinrich-Heine-Universität Düsseldorf

Sonderforschungsbereiche der Heinrich-Heine-Universität Düsseldorf

JÜRGEN KLÜNERS

Faktorisierung von Polynomen – Ein wichtiges Problem der Computeralgebra

Die Schwerpunkte meiner Forschungstätigkeiten sind in der Galois- und Zahlentheorie sowie in der Computeralgebra. In diesem Beitrag werde ich das Gebiet der Computeralgebra vorstellen, das auf der Schnittstelle zwischen Mathematik und Informatik liegt. Exemplarisch werde ich dann über aktuelle Fortschritte bei der Faktorisierung von Polynomen berichten.

Das wesentliche Ziel der Computeralgebra ist es, *Mathematik auf dem Computer* abzubilden. Hierzu werden weltweit so genannte Computeralgebrasysteme entwickelt, von denen Maple und Mathematica die bekanntesten sind. Wir können uns Computeralgebrasysteme als große Taschenrechner vorstellen, mit dem einzigen Unterschied, dass sie viel leistungsfähiger sind. So ist es zum Beispiel möglich, mit Zahlen zu rechnen, die aus mehreren Tausend Ziffern bestehen, oder Integrale oder Differentialgleichungen zu lösen. Weiterhin bieten die obigen Systeme die Möglichkeit, mathematische Funktionen zwei- oder sogar dreidimensional grafisch zu veranschaulichen. Computeralgebrapakete wie Maple und Mathematica werden mittlerweile in Schulen eingesetzt und finden auch Anwender außerhalb der Mathematik, zum Beispiel bei Natur- und Ingenieurwissenschaftlern.

Aus dem Versuch, Mathematik auf dem Computer abzubilden, haben sich in den vergangenen Jahren zwei Hauptrichtungen ausgebildet, die beide wichtig sind und verschiedene Anwendungen haben. In der so genannten *numerischen Mathematik* wird sehr viel mit reellen Approximationen gerechnet. Da wir reelle Zahlen auf einem Computer immer nur gerundet darstellen können, müssen wir hier sehr genau aufpassen, dass die mathematischen Fehler aufgrund von Rundungsproblemen nicht zu groß werden. Hierzu müssen die zu entwickelnden Algorithmen sehr genau angepasst und analysiert werden, um diese Effekte möglichst klein zu halten. Bei vielen Problemen, die aus der Anwendung kommen, erhalten wir die Daten bereits gerundet, da sie zum Beispiel als Messwerte von Versuchen kommen. Ein wichtiges Beispiel aus diesem Gebiet ist das numerische Lösen von Differentialgleichungen.

In diesem Bericht wollen wir stattdessen den *computeralgebraischen beziehungsweise symbolischen Zugang* verfolgen. Der wesentliche Unterschied zu numerischen Methoden besteht darin, dass wir versuchen, alles symbolisch zu berechnen. So ist für uns $\sqrt{2}$ ein Symbol, das quadriert 2 ergibt, und wir wollen vermeiden, mit der reellen Approximation $1,4142\ldots$ zu rechnen. Der offensichtliche Vorteil besteht darin, dass wir exakt rechnen und das deshalb keine Gefahr besteht, Fehler zu machen. Hierfür kaufen wir uns aber in vielen Fällen den Nachteil ein, dass die symbolischen Methoden langsamer als die entsprechenden numerischen sind. Wie gesagt hängt es von der expliziten Aufgabenstellung ab,

welcher der beiden Zugänge besser ist. In meinen Augen könnte es sehr interessant sein, wenn es zwischen diesen beiden Arbeitsrichtungen zu mehr Zusammenarbeit käme, da es sicherlich Probleme gibt, die man nur effizient lösen kann, wenn man sowohl numerische als auch symbolische Methoden anwendet. Dies werden wir aber in diesem Bericht nicht weiter verfolgen.

Betrachten wir ein sehr einfaches *Beispiel von symbolischem Lösen*, das als p-q-Formel bereits in der Schule vermittelt wird. Gegeben sei ein quadratisches Polynom

$$f(x) = x^2 + p \cdot x + q$$

mit Koeffizienten p und q. Dann sind die beiden Nullstellen gegeben durch:

$$-\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - q}.$$

Der Ausdruck unter der Wurzel ist auch wichtig für die Art der Nullstellen. So gibt es nur eine (das heißt eine doppelte) Nullstelle, wenn $(\frac{p}{2})^2 - q = 0$ gilt. Falls dieser Ausdruck echt größer als 0 ist, so haben wir zwei reelle Nullstellen, ansonsten können wir die beiden Nullstellen nur als komplexe Zahlen darstellen. Da in den meisten Fällen die komplexen Zahlen in der Schule nicht behandelt werden, gab es in diesem Fall keine Lösung, da wir (in der Schule) keine Wurzel aus einer negativen Zahl ziehen können.

Mit Hilfe der so genannten Cardano-Formeln können wir für Polynome bis zum Grad 4 die Nullstellen durch einen iterierten Wurzelausdruck bestimmen. Leider sehen diese Formeln insbesondere im Grad 4 nicht mehr so schön aus. Evariste Galois konnte dann im 19. Jahrhundert beweisen, dass solche Formeln im Allgemeinen ab Grad 5 nicht mehr existieren. Mit Hilfe der von ihm entwickelten Theorie können wir einem Polynom eine Gruppe zuordnen, die heutzutage *Galoisgruppe* genannt wird. Er konnte zeigen, dass die Nullstellen eines Polynoms genau dann durch iterierte Wurzelausdrücke beschrieben werden können, wenn die Galoisgruppe eine bestimmte Eigenschaft hat, nämlich eine auflösbare Gruppe ist. Bis zum Grad 4 sind alle Gruppen auflösbar, während es ab Grad 5 nicht auflösbare Gruppen gibt. In einem größeren Projekt in den vergangenen Jahren entwickelten wir einen Algorithmus zur Berechnung von Galoisgruppen und implementierten diesen auch in einem Computeralgebrasystem.[1] Zusammen mit Gunter Malle aus Kaiserslautern stellen wir auf unserer Homepage eine Datenbank zur Verfügung, die Polynome mit „interessanten“ Galoisgruppen enthält.[2]

Bevor wir zum Faktorisierungsproblem kommen, werden wir noch einige *grundlegende Probleme der Computeralgebra* erwähnen. Neben Langzahlarithmetiken ist es sehr wichtig, effiziente Algorithmen für fast alle Probleme aus der linearen Algebra zu haben. Beispielsweise sollten Polynome und Matrizen und die zugehörige Arithmetik effizient implementiert sein. Zur linearen Algebra gehört auch das effiziente Lösen von linearen Gleichungssystemen. Da viele Probleme nicht linear sind, beschäftigen wir uns in der Computeralgebra auch mit nicht linearen Gleichungssystemen. Für diese Art von Problemen gibt es im Allgemeinen nur Rechenverfahren, die eine exponentielle Laufzeit haben.

[1] Vgl. Fieker und Klüners (2008) sowie Geissler und Klüners (2000).

[2] Siehe auch Klüners und Malle (2001).

Dies bedeutet, dass die Laufzeit sehr stark wächst, wenn die Eingabedaten nur leicht vergrößert werden. Trotzdem hat es in diesem Bereich mit Hilfe von so genannten Gröbnerbasen erhebliche Fortschritte gegeben.

Der Autor dieses Berichts ist Mitentwickler der Computeralgebrapakete Kant und Magma, die in Berlin beziehungsweise Sydney entwickelt werden. Anwender dieser eher spezialisierten Pakete sind Mathematiker, die im Bereich der Algebra und Zahlentheorie arbeiten. In Magma sind auch die derzeit schnellsten bekannten Methoden zur *Faktorisierung von Zahlen* implementiert. Hier soll eine positive ganze Zahl in ihre Primfaktoren zerlegt werden; zum Beispiel lässt sich $60 = 2^2 \cdot 3 \cdot 5$ schreiben. Hierbei sind 2, 3 und 5 Primzahlen, das heißt Zahlen, die nur durch sich selbst und 1 teilbar sind. Viele Anwendungen aus der Kryptografie basieren darauf, dass das Faktorisieren von Zahlen ein schweres Problem ist. In dem so genannten RSA-Verfahren hängt die Sicherheit davon ab, dass eine jedem bekannte Zahl nicht in vertretbarer Zeit als Produkt von zwei Primzahlen geschrieben werden kann. So sind zum Beispiel die „sicheren“ Übertragungen im Internet mit dem RSA-Verfahren verschlüsselt. Auf den ersten Blick überraschend ist, dass das Faktorisieren von Polynomen in irreduzible (nicht zerlegbare) beziehungsweise Prim-Polynome deutlich einfacher als das entsprechende Problem für Zahlen ist. Wie wir später sehen werden, lassen sich zum Beispiel doppelte Faktoren bei Polynomen sehr einfach berechnen, während für das entsprechende Problem bei Zahlen kein effizienter Algorithmus bekannt ist.

Nehmen wir einmal an, dass wir daran interessiert sind, die Nullstellen des Polynoms $f(x) = x^4 - 2x^3 - 3x^2 + 4x - 1$ zu bestimmen. Über den ganzen Zahlen hat f die folgende Faktorisierung:

$$f(x) = x^4 - 2x^3 - 3x^2 + 4x - 1 = (x^2 - 3x + 1)(x^2 + x - 1).$$

Durch die Faktorisierung haben wir das Ursprungsproblem vom Grad 4 in zwei Probleme vom Grad 2 vereinfacht, die wir mit der p-q-Formel lösen können. Da das *Faktorisierungsproblem für Polynome* ein grundlegendes Problem der Computeralgebra ist, haben sich in den vergangenen Jahren sehr viele Wissenschaftler aus der Informatik und der Mathematik mit diesem Problem beschäftigt. Es wurden sehr viele Arbeiten veröffentlicht, in denen zum Beispiel über neue Ideen mit Laufzeitvorteilen von drei bis fünf Prozent berichtet wurde. So war es für die Fachwelt eine Riesenüberraschung, als 2002 Mark van Hoeij einen in der Praxis einsetzbaren Algorithmus veröffentlichte, der um Größenordnungen besser war als die bis dahin implementierten.[3] Er gab Beispiele an, die mit älteren Verfahren in mehreren Monaten nicht hatten gelöst werden können und die sein Programm in weniger als einer Minute berechnet hatte.

Wenn wir die Güte eines neuen Verfahrens analysieren wollen, so gibt es hierzu zwei Zugänge, die beide ihre Berechtigung haben. Im ersten Zugang messen wir die *Laufzeiten von echten Beispielen* und vergleichen diese mit den alten Implementierungen. Hier müssen wir dann darauf achten, dass beide Verfahren fair miteinander verglichen werden. So wäre es zum Beispiel unfair, wenn das eine Verfahren sehr liebevoll mit allen Tricks und das andere nur in seiner einfachsten Form implementiert ist. Ein anderes Problem besteht darin, geeignete Beispiele auszuwählen. So könnte es sein, dass ein gewisser Typ von

[3] Vgl. van Hoeij (2002).

Beispielen für das eine Verfahren günstiger ist, während das andere Verfahren bei anderen Beispielen besser ist. Für das Faktorisierungsproblem von Polynomen gibt es allgemein anerkannte Probleme (Challenges), die sich im Internet finden lassen, und die Implementierung von Mark van Hoeij war in allen Beispielen schneller. Wie bereits erwähnt gab es auch Beispiele, die erstmalig mit der neuen Implementierung gelöst werden konnten. Hierbei war es sogar so, dass die Implementierung des neuen Algorithmus im Gegensatz zu der Implementierung der bisherigen Verfahren noch nicht sehr ausgefeilt war. So bestand bei den Experten kein Zweifel, dass hier ein großer Durchbruch gelungen war. Innerhalb weniger Wochen wurde dann der neue Algorithmus in allen großen, teilweise auch kommerziellen, Computeralgebrapaketen implementiert.

Die zweite Möglichkeit, die Güte eines Verfahrens zu analysieren, besteht darin, theoretisch seine Laufzeit abzuschätzen. Hier landen wir dann im Arbeitsgebiet der *Komplexitätstheorie*, die auch auf der Schnittstelle zwischen Informatik und Mathematik liegt. Die Größe eines Polynoms ist bestimmt durch den Grad und die Größe seiner Koeffizienten. Da diese Koeffizienten Zahlen sind, können wir deren Größe durch die Anzahl ihrer Stellen messen. Nun versuchen wir, den Algorithmus zu analysieren, das heißt, wir möchten die Anzahl der Schritte zählen, die der Computer ausführen muss, um das gewünschte Ergebnis zu bestimmen. In der Komplexitätstheorie wird ein Algorithmus als effizient bezeichnet, wenn sich die Anzahl seiner Schritte als Polynom in der Größe der Eingabedaten beschreiben lässt. Zum Zeitpunkt, als Mark van Hoeij seinen neuen Algorithmus veröffentlichte, konnte er keine Laufzeitabschätzung im obigen Sinne angeben. Es war also nicht bekannt, ob der Algorithmus effizient ist. Dies war insbesondere deswegen unbefriedigend, da seit 1982 durch die berühmte LLL-Arbeit[4] bekannt war, dass man Polynome effizient, das heißt in Polynomlaufzeit, faktorisieren kann. Der zugehörige LLL-Algorithmus, der auch in dem neuen Verfahren von Mark van Hoeij verwendet wird, hat seinen Namen durch die drei Anfangsbuchstaben der Autoren bekommen.

In einer gemeinsamen Arbeit zusammen mit Mark van Hoeij und zwei weiteren Mathematikern aus Bordeaux und Sydney konnten wir die obige Lücke schließen.[5] Dies bedeutet, dass wir theoretisch beweisen konnten, dass der *neue van-Hoeij-Algorithmus* in Polynomzeit terminiert, das heißt effizient im obigen Sinne ist. Das bloße Vorhandensein einer solchen theoretischen Analyse bringt erst einmal nichts für die Geschwindigkeit einer eigentlichen Implementierung. Hier zeigte sich aber, dass das bessere Verständnis durch die genauere Analyse auch zu neuen Ideen führte, die wir zu besseren Implementierungen einsetzen konnten. Weiterhin haben wir gezeigt, dass die theoretischen Worst-Case-Laufzeiten stets besser sind als die in der Original-LLL-Arbeit bewiesenen Schranken. Wir haben zudem Beispiele gefunden, für die die Worst-Case-Zeiten des alten LLL-Algorithmus tatsächlich angenommen werden, während unsere Abschätzungen für den neuen Algorithmus immer noch sehr pessimistisch sind. Dies bedeutet, dass das Verfahren in der Praxis schneller fertig wird, als wir im schlimmsten Fall theoretisch annehmen. Zusammenfassend können wir nun sagen, dass der neue Algorithmus von Mark van Hoeij sowohl praktisch als auch theoretisch das beste bekannte Verfahren zur Polynomfaktorisierung ist.

[4] Vgl. Lenstra *et al.* (1982).
[5] Vgl. Belabas *et al.* (im Druck).

Damit wir das neue Verfahren von Mark van Hoeij erklären können, müssen wir zuerst das *Verfahren von Zassenhaus*[6] verstehen, das er 1969 entwickelt hat. Grob gesagt besteht die Idee dieses Verfahrens darin, dass wir unser Polynom über einem Körper faktorisieren, für den wir annehmen, dass dieses Problem einfacher ist. Zuerst werden wir dieses für den Körper der reellen Zahlen $\mathbb{R}$ erklären. Es ist bekannt, dass ein Polynom, das über den rationalen Zahlen $\mathbb{Q}$ definiert ist, über $\mathbb{R}$ in Linear- und quadratische Faktoren zerfällt. Über den rationalen Zahlen kann es irreduzible Faktoren von deutlich höherem Grad geben. Ein irreduzibler Faktor über $\mathbb{Q}$ zerfällt dann über $\mathbb{R}$ in viele Faktoren vom Grad 1 oder 2. Im Prinzip ist unser Problem gelöst, wenn wir die richtigen Faktoren über $\mathbb{R}$ finden und dann ausmultiplizieren. Mit anderen Worten beruht der Zassenhaus-Algorithmus auf einem geschickten Probierverfahren, das dann sehr effizient ist, wenn wir wenig ausprobieren müssen. Um dieses Probieren besser zu erläutern, betrachten wir folgendes Beispiel:

$$f(x) = x^6 - x^5 - 5x^4 + 3x^3 + 7x^2 - x - 1 \in \mathbb{Z}[x].$$

Dieses Polynom zerfällt über $\mathbb{R}$ in sechs Linearfaktoren und die approximierten Nullstellen sehen so aus:

$$0.44504, 1.80193, -1.24697, -0.34729, -1.53208, 1.87938.$$

Wir sehen sofort, dass keine dieser Nullstellen in den ganzen Zahlen $\mathbb{Z}$ liegt. Nach einem Satz von Gauß wissen wir, dass ein normierter Faktor von f, der Koeffizienten in den rationalen Zahlen besitzt, bereits Koeffizienten in $\mathbb{Z}$ haben muss. Wir müssen uns an dieser Stelle klarmachen, dass wir nur mit reellen Approximationen gerechnet haben. Die obigen Zahlen sind aber so weit von einer ganzen Zahl entfernt, dass unsere Methoden ausreichen, um zu zeigen, dass sie nicht ganz sind. Was hätten wir aber gemacht, wenn das Ergebnis beispielsweise 2.00001 gewesen wäre? Ist dies nun gleich 2 oder nicht? An diesem Beispiel sehen wir, dass es einfacher zu zeigen ist, dass eine Zahl nicht ganz ist, als zu zeigen, dass sie ganz ist. Beim Faktorisierungsproblem können wir uns aber einfach aus der Affäre ziehen, indem wir testen, ob $x-2$ unser Polynom teilt.

Nun fängt das eigentlich *Probierverfahren* an. Wir haben gesehen, dass es keine linearen Faktoren gibt. Nun müssen wir testen, ob es quadratische Faktoren gibt. Insgesamt müssen wir $15 = \binom{6}{2}$ Möglichkeiten testen. Wir geben aus Platzgründen nur die fünf Varianten an, bei denen der Faktor $x - 0.44504$ beteiligt ist:

$$x^2 - 2.246x + 0.801, x^2 + 0.801x - 0.554, x^2 - 0.0977x - 0.154,$$

$$x^2 + 1.087x - 0.681, x^2 - 2.324x + 0.836.$$

Wir sehen, dass keines dieser Polynome ganze Koeffizienten hat. Gleiches gilt für die anderen zehn Möglichkeiten. Nun testen wir Faktoren vom Grad 3. Hier gibt es $\binom{6}{3} = 20$ Möglichkeiten. Wir geben hier drei von diesen an:

$$x^3 - 2.692x^2 + 1.801x - 0.356, x^3 - 4.048x^2 + 4.850x - 1.445,$$

$$x^3 - x^2 - 1.999x + 0.999.$$

[6] Vgl. Zassenhaus (1969).

Die ersten beiden Möglichkeiten korrespondieren nicht zu Polynomen mit ganzen Koeffizienten, während aber das dritte Beispiel ganz vielversprechend aussieht. Wir testen nun mit Hilfe einer Probedivision, ob x^3-x^2-2x+1 ein Teiler von f ist und erhalten eine positive Antwort:

$$(x^6-x^5-5x^4+3x^3+7x^2-x-1):(x^3-x^2-2x+1)=x^3-3x-1.$$

Damit ist im Prinzip die *Idee des Zassenhaus-Algorithmus* bereits gut beschrieben. Wenn m die Anzahl der Faktoren über den reellen Zahlen bezeichnet, müssen wir im schlimmsten Fall 2^m Möglichkeiten testen. Da m immer mindestens $n/2$ ist, wobei n den Grad unseres Polynoms bezeichnet, sehen wir, dass dies recht schnell zu viele Möglichkeiten werden. Zassenhaus hatte in seiner Arbeit vorgeschlagen, anstatt reeller Approximationen die Approximationen modulo p^k zu verwenden, wobei p eine Primzahl und k eine positive ganze Zahl ist. Vornehm gesprochen werden die reellen Zahlen durch die so genannten p-adischen Zahlen ersetzt. Dies bietet in der Praxis zwei Vorteile. Erstens sind Fehler bei p-adischen Approximationen besser kontrollierbar als die entsprechenden Fehler bei reellen Zahlen. Weiterhin können irreduzible modulo p^k-Faktoren einen höheren Grad haben. In vielen praktischen Beispielen ist daher die Anzahl m der modulo p^k-Faktoren deutlich kleiner als der Grad n des gegebenen Polynoms. In diesen Fällen entfaltet dann der Zassenhaus-Algorithmus seine volle Stärke, da er nicht zu viele Möglichkeiten testen muss. Ein weiterer Vorteil der Approximationen modulo p^k ist, dass wir uns die Primzahl aussuchen können. Bei den reellen Zahlen müssen wir damit leben, was wir bekommen. So können wir hoffen, dass wir immer eine gute Primzahl, das heißt eine mit kleinem m, finden können. Leider gibt es Beispiele, wo dies nicht möglich ist. Für diese Beispiele bringt der neue Algorithmus von van Hoeij drastische Verbesserungen, während er sich in den anderen Varianten ähnlich gut wie der Zassenhaus-Algorithmus verhält.

Für das weitere Verständnis erklären wir kurz, was *endliche Körper* und Approximationen modulo p^k sind. Hierzu sei p eine Primzahl. Dann können wir den endlichen Körper $\mathbb{F}_p$ mit p Elementen so interpretieren, dass er aus den Zahlen $\{0,1,\ldots,p-1\}$ besteht. Wenn wir nun zwei Zahlen addieren beziehungsweise multiplizieren, so verwenden wir die normale Addition beziehungsweise Multiplikation. Wenn das Ergebnis nicht in $\{0,1,\ldots,p-1\}$ liegt, so reduzieren wir das Ergebnis, indem wir den Rest aus der Division mit p bestimmen. Dieser Rest liegt dann zwischen 0 und $p-1$. Wir können leicht zeigen, dass $\mathbb{F}_p$ mit diesen Operationen ein Körper ist, wobei es wesentlich ist, dass p eine Primzahl ist.

Die *Approximationen modulo* p^k können wir ganz ähnlich erklären. Wir betrachten die Menge $\{0,1,\ldots,p^k-1\}$ und erklären Addition und Multiplikation wie bei endlichen Körpern, mit dem Unterschied, dass wir am Ende den Rest der Division mit p^k bestimmen. Was wir erhalten, ist nur noch ein Ring und kein Körper. Beispielsweise erhalten wir für $p^k=2^2=4$ den Effekt, dass $2\cdot 2=0$ ist. Die Menge $\{0,1,\ldots,p^k-1\}$ ist eine häufige Wahl, aber das so genannte symmetrische Restsystem $\{(-p^k+1)/2,\ldots,(p^k-1)/2\}$ ist für unsere Anwendungen die bessere Wahl. Die Arithmetik funktioniert auf dieselbe Weise – mit dem einzigen Unterschied, dass wir am Ende noch einmal p^k abziehen, wenn die Division mit Rest etwas liefert, was größer als $p^k/2$ ist. Der Vorteil des symmetrischen Restsystems liegt darin, dass es besser mit negativen Zahlen harmoniert. So ist zum Beispiel die negative Zahl -5 immer -5 modulo 11 oder modulo 11^2 oder sogar modulo 11^k

für alle $k \in \mathbb{N}$. Hier wird auch deutlich, warum wir von einer Approximation sprechen. Wenn wir genau genug approximieren, das heißt k groß genug wählen, erhalten wir das gewünschte Ergebnis und es bleibt erhalten, wenn wir noch genauer werden, das heißt den Exponenten k größer machen.

Wir erklären jetzt noch einmal den *Zassenhaus-Algorithmus*, wobei wir diesmal die Approximationen modulo p^k verwenden. Dies ist die Form, in der dieser Algorithmus tatsächlich implementiert ist. Wir können annehmen, dass unser Polynom f normiert und quadratfrei ist, also keine mehrfachen Faktoren hat, da diese den größten gemeinsamen Teiler von f und der Ableitung f' teilen, den wir effizient berechnen können. Im Zassenhaus-Algorithmus wählen wir eine Primzahl p so, dass f modulo p keine doppelten Faktoren besitzt. Hierbei können wir jede Primzahl wählen, die nicht die Diskriminante von f teilt. Wir bezeichnen mit $\bar{f} \in \mathbb{F}_p[x]$ dasjenige Polynom, das aus f dadurch entsteht, dass wir jeden Koeffizienten modulo p reduzieren. Mit bekannten Algorithmen[7] erhalten wir eine Faktorisierung der folgenden Form:

$$\bar{f}(x) = \bar{f}_1(x) \cdots \bar{f}_r(x) \in \mathbb{F}_p[x].$$

Wir merken an, dass das Berechnen dieser Faktorisierung für kleine Primzahlen äußerst effizient ist und nur einen Bruchteil der Gesamtlaufzeit des Algorithmus ausmacht. Mittels des so genannten Hensel-Liftings können wir sehr effizient (durch Lösen von linearen Gleichungssystemen) für jedes $k \in \mathbb{N}$ eine Faktorisierung der folgenden Form bestimmen:[8]

$$f(x) \equiv \tilde{f}_1(x) \cdots \tilde{f}_r(x) \bmod p^k,$$

wobei (bei geeigneter Einbettung) $\tilde{f}_i \equiv \bar{f}_i \bmod p$ gilt. Wir erklären nun an dem Beispiel $f(x) = x^4 - 11$ den Zassenhaus-Algorithmus. Für $p = 13$ erhalten wir, dass $\bar{f} \in \mathbb{F}_{13}[x]$ irreduzibel ist. Hieraus folgt dann sofort die Irreduziblität von $f \in \mathbb{Z}[x]$ und wir sehen die Überlegenheit im Vergleich zu den reellen Approximationen. Damit wir das Probierverfahren anhand eines kleinen Beispiels erklären können, wählen wir stattdessen (eine schlechte Wahl!) $p = 5$ und erhalten modulo 5 nur Linearfaktoren, die wir mittels Hensel-Lifting (sehr schnell!) wie folgt liften:

$$f(x) \equiv (x+41)(x-38)(x+38)(x-41) \bmod 125.$$

Wir können nun mit dem folgenden Satz abschätzen, dass die Koeffizienten eines Faktors betraglich kleiner als 33 sind.[9]

Satz 1 (Landau-Mignotte)**.** *Sei g ein Faktor eines normierten Polynoms $f \in \mathbb{Z}[x]$ mit*

$$f(x) = \sum_{i=0}^{n} a_i x^i \text{ und } g(x) = \sum_{i=0}^{m} b_i x^i.$$

Dann gilt: $|b_i| \leq \binom{m}{i} ||f||_2$, wobei $||f||_2 := \sqrt{\sum_{i=0}^{n} a_i^2}$ die 2-Norm ist.

[7] Vgl. von zur Gathen und Gerhard (1999), Kapitel 14.
[8] Siehe zum Beispiel von zur Gathen und Gerhard (1999), Kapitel 15.
[9] Siehe von zur Gathen und Gerhard (1999: 155ff.).

Somit kann f keinen Linearfaktor besitzen, da alle Linearfaktoren modulo 125 im symmetrischen Restsystem $\{-62,\ldots,62\}$ einen Koeffizienten größer als 33 besitzen. Als Nächstes müssen wir probieren, ob das Produkt von zwei mod-125-Faktoren von f zu einem echten Faktor korrespondiert.

$$\begin{aligned}(x+41)(x-38) &\equiv x^2+3x-58 \mod 125,\\ (x+41)(x+38) &\equiv x^2-46x+58 \mod 125,\\ (x+41)(x-41) &\equiv x^2-56 \mod 125.\end{aligned}$$

Wieder enthält jedes dieser quadratischen Polynome wenigstens einen Koeffizienten, der betraglich größer ist als unsere obere Schranke 33. Damit kann der „modulare Faktor" $(x+41)$ kein Teiler von einem linearen oder quadratischem Polynom $g \in \mathbb{Z}[x]$ sein mit $g \mid f \in \mathbb{Z}[x]$. Wir haben also wieder gezeigt, dass das Polynom f irreduzibel ist. Wenn unser gegebenes Polynom reduzibel ist, so finden wir bei unserer Suche modulare Faktoren, deren Koeffizienten kleiner als die berechnete Schranke sind. Für diese Polynome testen wir mit einer Probedivision, ob es sich tatsächlich um einen Faktor handelt. Da die Probedivisionen teuer sind, sollte p^k deutlich größer als die berechnete Schranke gewählt werden, damit falsche Kandidaten eine hohe Wahrscheinlichkeit auf zu große Koeffizienten haben.

Die Wahl von $p = 5$ im obigen Beispiel war recht künstlich, aber es gibt irreduzible Polynome, bei denen immer sehr viele „modulare Faktoren" auftreten. Wenn wir zum Beispiel ein Polynom vom Grad 2^n mit elementarabelscher Galoisgruppe $(\mathbb{Z}/2\mathbb{Z})^n$ wählen, so ist dieses irreduzibel, besitzt aber für jede Primzahl mindestens 2^{n-1} modulare Faktoren. Bei der *Analyse des Zassenhaus-Algorithmus* stellen wir fest, dass die meisten Teile des Algorithmus sehr effizient sind. Da wir eine kleine Primzahl p wählen können, stellt die Faktorisierung von $\bar{f} \in \mathbb{F}_p[x]$ kein Problem dar. Auch das Hensel-Lifting kann sehr effizient implementiert werden. Der Flaschenhals entsteht dann, wenn die Anzahl r der modularen Faktoren groß ist, da wir dann im Wesentlichen 2^r Tests durchführen müssen. Um einen effizienten Algorithmus zu bekommen, müssen wir die Anzahl dieser Tests drastisch reduzieren.

An dieser Stelle steigt der neue *van-Hoeij-Algorithmus* ein. Einerseits können wir mittels einer leichten Variation (siehe später die Abbildung Φ) der logarithmischen Ableitung das multiplikative Problem in ein additives (lineares) Problem verwandeln. In einem zweiten Schritt wird dann das kombinatorische Suchproblem auf ein so genanntes Rucksackproblem reduziert. Normalerweise sind Rucksackprobleme auch sehr schwer lösbar, aber dieses spezielle kann sehr geschickt durch den Einsatz des LLL-Gitterreduktionsalgorithmus gelöst werden. Wir erinnern uns, dass dieser Reduktionsalgorithmus eingeführt wurde, um zu zeigen, dass die Polynomfaktorisierung in polynomieller Laufzeit gelöst werden kann. Zur Lösung des oben angesprochenen Rucksackproblems werden aber andere Gitter verwendet. In der Originalarbeit[10] bestand die Hauptidee darin, die Koeffizienten eines Faktors zu bestimmen. Im Rucksackproblem wollen wir die richtigen 0-1-Vektoren finden (wird später noch erklärt). Der große Fortschritt von dem neuen Algorithmus besteht also darin, dass wir sehr kleine Vektoren in geeigneten Gittern suchen. Wir fixieren die folgende Notation:

[10] Vgl. Lenstra *et al.* (1982).

$$f = g_1 \cdots g_s \in \mathbb{Z}[x] \text{ und } f = \tilde{f}_1 \cdots \tilde{f}_r \in \mathbb{Z}_p[x].$$

Die Faktorisierung über den p-adischen Zahlen $\mathbb{Z}_p$ können wir algorithmisch immer nur modulo p^k bestimmen, wobei wir $k \in \mathbb{N}$ geeignet wählen. Für das Verständnis des Folgenden reicht es aus, sich die p-adischen Zahlen $\mathbb{Z}_p$ als Approximationen modulo p^k für ein beliebig großes k vorzustellen. Wir bezeichnen dann mit $\mathbb{Q}_p$ den Quotientenkörper, das heißt die Menge, die aus Brüchen p-adischer Zahlen besteht. Nun schreiben wir:

$$g_v := \prod_{i=1}^{r} f_i^{v_i} \text{ für } v = (v_1, \ldots, v_r) \in \{0,1\}^r$$

und erhalten als neues *Problem*: Für welche $v \in \{0,1\}^r$ gilt: $g_v \in \mathbb{Z}[x]$?

Um dieses Problem effizient zu lösen, ist es wesentlich, dass wir es linearisieren, das heißt aus unserem multiplikativen Problem ein additives machen. Hierzu betrachten wir im Wesentlichen die *logarithmische Ableitung*, wobei $\mathbb{Q}_p(x) := \{\frac{a(x)}{b(x)} \mid a, b \in \mathbb{Q}_p[x]\}$:

$$\Phi : \mathbb{Q}_p(x)^* / \mathbb{Q}_p^* \to \mathbb{Q}_p(x), g \mapsto \frac{fg'}{g}.$$

Es ist sofort klar, dass Φ additiv ist, das heißt $\Phi(g_{v_1}) + \Phi(g_{v_2}) = \Phi(g_{v_1+v_2})$. Weiterhin gilt für beliebige $v \in \mathbb{Z}^r$, dass $\Phi(g_v)$ ein Polynom ist, also in $\mathbb{Z}_p[x]$ liegt.

Jetzt fehlt noch ein kleiner Schritt zur *Übersetzung auf ein gittertheoretisches Problem*. Wir definieren Vektoren $w_1, \ldots, w_s \in \{0,1\}^r$ derart, dass für die echten Faktoren $g_1, \ldots, g_s \in \mathbb{Z}[x]$

$$g_i = \prod_{1 \le j \le r} \tilde{f}_j^{w_{i_j}}$$

gilt. Diese erzeugen ein Gitter $W = \langle w_1, \ldots, w_s \rangle \subseteq \mathbb{Z}^r$, wobei letzteres Gitter von den Einheitsvektoren erzeugt wird, die zu den Faktoren $\tilde{f}_i$ korrespondieren. Wesentlich für das nun folgende Verfahren ist die Eigenschaft, dass ein $v \in \mathbb{Z}^r$ genau dann in W liegt, wenn $\Phi(g_v) \in \mathbb{Z}[x]$ gilt. Wenn wir nun ein beliebiges Erzeugendensystem von W kennen, so können wir die „kanonischen" Basisvektoren $w_1, \ldots, w_s$ sehr einfach mit Methoden der linearen Algebra oder einfach kombinatorisch bestimmen. Kennen wir die w_i und ist p^k größer als das Doppelte der Landau-Mignotte-Schranke, so können wir die Faktoren g_i wie im Zassenhaus-Algorithmus rekonstruieren.

Die *Idee des Algorithmus* ist wie folgt: Wir starten mit dem Gitter $L = \mathbb{Z}^r$ und wissen $W \subseteq L$. Nun konstruieren wir ein Untergitter $L' \subset L$, das aber immer noch W enthalten soll. Die Hoffnung ist, dass wir nach endlich vielen Iterationen bei $L' = W$ landen. Wir merken an, dass wir diese Situation sehr einfach testen können. Wenn unser Normalform-Algorithmus eine Basis aus 0-1-Vektoren liefert und diese zu Faktoren aus $\mathbb{Z}[x]$ korrespondieren (Test durch Probedivision), dann gilt $L' = W$.

An dieser Stelle betrachten wir ein verwandtes Problem, nämlich die *Faktorisierung von bivariaten Polynomen* über einem endlichen Körper. Bei diesem Problem wollen wir ein gegebenes Polynom $f \in \mathbb{F}_p[t][x]$ faktorisieren. Auch wenn diese Situation auf den ersten

Blick etwas schwieriger zu verstehen ist, so stellt sich heraus, dass dieses Problem deutlich einfacher zu lösen ist. Die Landau-Mignotte-Schranke vereinfacht sich zu

$$g \mid f \in \mathbb{F}_p[t][x] \Rightarrow \deg_t(g) \leq \deg_t(f),$$

wobei $\deg_t(f)$ der t-Grad des Polynoms f ist. Der Einfachheit halber gehen wir davon aus, dass $\bar{f}(x) := f(0,x) \in \mathbb{F}_p[x]$ quadratfrei ist. Dies ist eine echte Einschränkung, da es sein kann, dass $f(a,x)$ doppelte Faktoren für alle $a \in \mathbb{F}_p$ besitzt. Zur Lösung dieses Problems verweisen wir auf Belabas *et al.* (im Druck). Aus der Faktorisierung $\bar{f} = \bar{f}_1 \cdots \bar{f}_r \in \mathbb{F}_p[x]$ erhalten wir mittels Hensel-Lifting eine Faktorisierung $f(t,x) = \tilde{f}_1 \cdots \tilde{f}_r$ im Potenzreihenring $\mathbb{F}_p[[t]][x]$, die wir in der Praxis modulo t^k approximieren können. Die Funktion Φ ist nun wie folgt definiert:

$$\Phi : \mathbb{F}_p[[t]](x)^*/\mathbb{F}_p[[t]](x^p)^* \to \mathbb{F}_p[[t]](x), g \mapsto \frac{fg'}{g}.$$

Zur Situation über $\mathbb{Q}$ besteht der kleine Unterschied, dass die Ableitung von x^p null ist. Die Gitter L und W werden analog zur Situation in $\mathbb{Z}[x]$ definiert. Sei nun $v \in L \setminus W$. Dann gilt:

$$\mathrm{Pol}(v) := \Phi(g_v)(x) = \sum_{i=1}^{r} v_i \Phi(f_i) =$$

$$\sum_{i=0}^{n-1} b_i x^i \in \mathbb{F}_p[[t]][x] \setminus \mathbb{F}_p[t][x].$$

Weiterhin gilt für $g_v \in \mathbb{F}_p[t][x]$ die Abschätzung $\deg_t(b_i) \leq \deg_t(f)$. Wir wählen nun ein $k > \deg_t(f)$ und bestimmen für $v \in L$ das zugehörige Polynom

$$g_v \equiv \sum_{i=0}^{n-1} b_i(t) x^i \bmod t^k,$$

wobei wir kanonisch die Polynome $b_i(t)$ modulo t^k reduzieren. Falls nun eines dieser Polynome b_i einen t-Grad hat, der größer als $\deg_t(f)$ ist, so wissen wir, dass das zugehörige v nicht in W liegen kann. Im Folgenden vermeiden wir den kombinatorischen Ansatz. Hierzu bezeichnen wir mit $e_1, \ldots, e_r \in \mathbb{F}_p^r$ die Standardbasis von $\mathbb{F}_p^r$ und identifizieren die Elemente von $\mathbb{F}_p$ mit $\{0, \ldots, p-1\}$. Wir definieren $m := \deg_t(f)$ und

$$A_i := \begin{pmatrix} b_{i,m,1} & \cdots & b_{i,m,r} \\ b_{i,m+1,1} & \cdots & b_{i,m+1,r} \\ \vdots & \ddots & \vdots \\ b_{i,k-1,1} & \cdots & b_{i,k-1,r} \end{pmatrix} \in \mathbb{F}_p^{(k-m)\times r},$$

wobei die $b_{i,j,\ell}$ durch

$$\mathrm{Pol}(e_\ell) \equiv \sum_{i=0}^{n-1} \sum_{j=0}^{k-1} b_{i,j,\ell} t^j x^j \bmod t^k \quad (1 \leq \ell \leq r)$$

gegeben sind. Nun erfüllen alle $v \in W$ die Gleichung $A_i v^{\mathrm{tr}} = 0$. Wir können also durch sukzessive Kernberechnung immer kleinere Gitter L' bestimmen, die immer noch W enthalten. In einer leicht verbesserten Version dieses Algorithmus zeigen wir in Belabas *et al.* (im Druck), dass schließlich $L' = W$ gilt:

Satz 2. *Sei* $k > (2n-1)\deg_t(f)$. *Dann ist* W *der Kern von* $A_1, \ldots, A_{n-1}$.

Im Folgenden geben wir eine kurze Beweisskizze für diesen Satz an. Sei $v \in L \setminus W$ so gewählt, dass g_v keine p-te Potenz ist. Dann können wir v mit Hilfe der $w_1, \ldots, w_s$ so abändern, dass Folgendes gilt:

1. $f_i \mid \mathrm{Pol}(v)$ für ein $1 \le i \le r$.
2. $g_j \nmid \mathrm{Pol}(v)$ für alle $1 \le j \le s$.

Damit gilt aber für dieses neue v und $H := \mathrm{Pol}(v) \bmod t^k$ (aufgefasst als Polynom in $\mathbb{F}_p[t][x]$):

$$\mathrm{Res}(f, \mathrm{Pol}(v)) = 0 \text{ und } \mathrm{Res}(f, H) \neq 0.$$

Also gilt $t^k \mid \mathrm{Res}(f,H)$, was bei genügend großem k einen Widerspruch zur Definition der Resultante durch die Sylvester-Matrix bedeutet.

Kommen wir nun zurück zu unserem *Faktorisierungsproblem über* $\mathbb{Z}$. Hier können wir nicht exakt denselben Algorithmus anwenden, da es beim Addieren Überträge gibt, zum Beispiel $(3+1\cdot 5^1)+(3+1\cdot 5^1) = (1+3\cdot 5^1) \neq 1+2\cdot 5^1$ in $\mathbb{Z}_5$. Allerdings können wir zeigen, dass die Fehler durch Überträge klein sind. Anstatt des linearen Gleichungssystems verwenden wir nun ein geeignetes Gitter und suchen nach Vektoren kurzer Länge. Da das Finden von kürzesten Vektoren in allgemeinen Gittern NP-vollständig (also sehr schwer) ist, ist es wesentlich, die Gitter geeignet zu wählen. Für diese Gitter verwenden wir dann die LLL-Gitterreduktion und können garantieren, dass die ersten Basisvektoren unseres Gitters gerade eine Darstellung von W liefern. Wir benutzen die analogen Notationen wie im bivariaten Fall und erhalten:

$$\mathrm{Pol}(e_\ell) \equiv \sum_{i=0}^{n-1} b_{i,\ell} x^i \bmod p^k \quad (1 \le \ell \le r).$$

Wir definieren nun ein Gitter Λ, das durch die Spalten der folgenden Matrix definiert wird:

$$A := \begin{pmatrix} I_r & 0 \\ \tilde{A} & p^k I_n \end{pmatrix} \text{ mit } \tilde{A} := \begin{pmatrix} b_{0,1} & \cdots & b_{0,r} \\ \vdots & \ddots & \vdots \\ b_{n-1,1} & \cdots & b_{n-1,r} \end{pmatrix}.$$

Wenn wir einen Vektor aus Λ auf die ersten r Zeilen projizieren, erhalten wir einen Vektor aus L. Wenn wir die Präzision p^k groß genug wählen, dann können wir beweisen, dass alle Vektoren aus Λ, deren letzte n Einträge kleiner als die Landau-Mignotte-Schranke sind, zu einem Vektor aus W korrespondieren. Wir werden also obiges Gitter LLL-reduzieren und können dann beweisen, dass die ersten s Vektoren W erzeugen. Wir können sehr einfach eine obere Schranke B für die Norm der zu $w_1, \ldots, w_s$ gehörigen Gittervektoren in Λ berechnen. Für den LLL-Ansatz ist dann folgendes Lemma sehr nützlich, da es bei zu niedrig gewählter Präzision p^k immerhin erlaubt, einen Fortschritt zu erzielen, das heißt ein Untergitter $L' \subset L$ zu berechnen, das immer noch W enthält.

Lemma 1. *Sei Λ ein Gitter mit Basis $b_1, \ldots, b_m$ und Gram-Schmidt-Basis $b_1^*, \ldots, b_m^*$. Definiere $t := \min\{i \mid \forall i < j \leq m : ||b_j^*||_2 > B\}$. Dann sind alle Vektoren b mit $||b||_2 \leq B$ bereits in $\mathbb{Z}b_1 + \ldots + \mathbb{Z}b_t$ enthalten.*

Analog zum bivariaten Fall brauchen wir nun noch eine Präzisionsabschätzung, wann wir garantiert fertig sind. Wenn wir diese Präzision p^k verwenden, terminiert unser Algorithmus in einem Schritt und wir erhalten dann die polynomielle Laufzeit. Wir merken an, dass diese Abschätzung in der Praxis nicht benötigt wird, da wir dort so lange die Präzision erhöhen, bis wir fertig sind.

Satz 3. *Sei $f \in \mathbb{Z}[X]$ vom Grad n. Dann terminiert der oben beschriebene Algorithmus, wenn*

$$p^k > c^n \cdot 4^{n^2} ||f||_2^{2n-1}$$

gilt, wobei c eine explizit berechenbare Konstante ist.

Wenn wir eine Laufzeitabschätzung für diesen Algorithmus bestimmen, so bekommen wir dieselbe Laufzeit wie in der Original-LLL-Arbeit. Der wesentliche Unterschied ist, dass wir Beispiele angeben können (zum Beispiel irreduzible Polynome), bei denen diese Laufzeit im Original-LLL-Ansatz tatsächlich auftritt. In der Praxis zeigt sich für unseren Algorithmus, dass die abgeschätzte Laufzeit äußerst pessimistisch ist und wir kennen kein Beispiel, in denen diese annähernd erreicht wird. In der Arbeit Belabas *et al.* (im Druck) geben wir noch eine Variante dieses Algorithmus an, in der wir die Gitterreduktion sukzessive nur mit kleinen Einträgen durchführen. Dies ergibt zum einen eine theoretische Laufzeitverbesserung, aber auch in der Praxis zeigen sich deutliche Vorteile.

Literatur

BELABAS, K., M. VAN HOEIJ, J. KLÜNERS und A. STEEL (im Druck). „Factoring polynomials over global fields“, *Journal de Théorie des Nombres de Bordeaux*.

FIEKER, C. und J. KLÜNERS (2008). *On computing Galois groups*. Preprint.

VON ZUR GATHEN, J. und J. GERHARD (1999). *Modern computer algebra*. Cambridge.

GEISSLER, K. und J. KLÜNERS (2000). „Galois group computation for rational polynomials“, *Journal of Symbolic Computation* 30, 653–674.

VAN HOEIJ, M. (2002). „Factoring polynomials and the knapsack problem“, *Journal of Number Theory* 95, 167–189.

KLÜNERS, J. und G. MALLE (2001). „A Database for field extensions of the rationals“, *LMS Journal of Computation and Mathematics* 4, 182–196.

LENSTRA, A. K., H. W. LENSTRA, JR. und L. LOVÁSZ (1982). „Factoring polynomials with rational coefficients“, *Mathematische Annalen* 261 (4), 515–534.

ZASSENHAUS, H. (1969). „On Hensel factorization I“, *Journal of Number Theory* 1, 291–311.

ISBN 978-3-940671-10-3