Vom Fachbereich Mathematik der Technischen Universität Kaiserslautern
zur Verleihung des akademischen Grades Doktor der Naturwissenschaften
(Doctor rerum naturalium, Dr. rer. nat.) genehmigte Dissertation

Factorization of multivariate polynomials

Martin Mok-Don Lee

June 10, 2013

1. Gutachter: Prof. Dr. Gerhard Pfister
2. Gutachter: Prof. Dr. Dorin-Mihail Popescu

D 386

Contents

List of Tables

Acknowledgements

Danke an alle die mich während der Zeit meiner Promotion unterstützt haben, besonders meine Eltern. Für meine Nichte Sarah.

Financial Support

This work was funded by Stipendienstiftung des Landes Rheinland-Pfalz and DFG Priority Project SPP 1489.

Chapter 1

Introduction

Multivariate polynomial factorization is a cornerstone of many applications in computer algebra. It is used in computing primary decompositions of ideals, Gröbner basis, and many other applications. On the other hand, it can serve as a benchmark for many basic operations such as univariate polynomial factorization, polynomial multiplication, computation of greatest common divisors, and various others, as these operations are extensively used in almost all known algorithms. Last but not least, multivariate polynomial factorization is a challenge in itself.

Its beginnings in modern mathematics can be traced back to Zassenhaus [Zas69]. He first described an algorithm to factorize univariate polynomials over $\mathbb{Z}$, whereas Musser [Mus71] generalized it to the multivariate case. Musser also states a general algorithm whose outline looks like this:

1. remove multiple factors
2. reduce to univariate problem by plugging in a suitable point
3. factorize the resulting univariate poly
4. reconstruct tentative multivariate factors from univariate factors by Hensel lifting
5. recombine tentative factors to obtain true factors

An algorithm of this kind is also referred to as Extended Zassenhaus (EZ) algorithm.

In the course of this thesis we address each step of the algorithm and the problems that arise therein, and always focus on a fast implementation of

each step. Furthermore, we restrict ourselves to the coefficient rings $\mathbb{F}_q$, $\mathbb{Z}$, and $\mathbb{Q}(\alpha)$.

During the past 40 years numerous improvements, variants, and generalizations of the original outline and its underlying steps have been made. For a survey and historic context we refer the reader to [Kal82], [Kal90], [Kal92], [Kal03]; a more recent survey can be found in [BL12].

Our contribution is a new bound on the coefficients of a factor of a multivariate polynomial over $\mathbb{Q}(\alpha)$. The new bound does not require α to be an algebraic integer, and only depends on the height of the polynomial and the height of the minimal polynomial. This bound is used to design an Extended Zassenhaus algorithm which computes the Hensel lift of the univariate factors modulo some high enough power of a prime. A similar idea is used by Lenstra [Len84] requiring α to be an algebraic integer, but it is impractical due to reduction of large lattices; or in [JM09] in the more general setting of algebraic function fields, but only with a heuristic bound. We compare an implementation of the latter in `Maple` to our algorithm in 11.3.5.

Furthermore, we adapt a method by Kaltofen [Kal85c] to precompute the leading coefficients of multivariate factors and enhance it by various heuristics to obtain a smaller multiplier.

Moreover, we improve the complexity of Bernardin's Hensel lifting algorithm [Ber99] by a constant factor.

Almost all algorithms mentioned here were implemented by the author in the `C++` library `factory`, which is part of the computer algebra system `Singular` [DGPS12]. To the best of our knowledge, our implementation is the first open-source implementation of [BvHKS09] and the only open-source implementation that is able to factorize polynomials over such a range of coefficient domains at such speed.

Chapter 2

Preliminaries

All rings considered here are commutative with 1.

Since a thorough introduction to the rich field of algebra is beyond the scope of this thesis, we restrict ourselves to some very basic definitions and facts that are used throughout this text. A short summary of concepts of univariate polynomial rings and finite fields can be found in [Lee09]. For an extensive introduction, we refer the reader to [Bos06] or any other introductory book on this topic.

Definition 2.1. An n-variate polynomial over a ring R is a sequence $(a_\alpha)_{\alpha\in\mathbb{N}^n}$, where $a_\alpha \in R$ for all $\alpha \in \mathbb{N}^n$ and only finitely many a_α are non-zero. Usually a polynomial is represented by $\sum_{|\alpha|=0}^{d} a_\alpha x_1^{\alpha_1} \cdot \ldots \cdot x_n^{\alpha_n}$, where $|\alpha| = \sum_{i=1}^{n} \alpha_i$. With the rules for adding and multiplying sums, the set of polynomials over R becomes a ring, denoted by $R[x_1, \ldots, x_n] = R[\mathbf{x}]$. For a polynomial $f = \sum_{|\alpha|=0}^{d} a_\alpha \mathbf{x}^\alpha \in R[\mathbf{x}]$, which is not the zero polynomial, we define the following:

- $\operatorname{tdeg}(f) = \max\{|\alpha| \mid a_\alpha \neq 0\}$, the total degree of f.

The above definition leads to a distributive representation of a polynomial but one can also consider the recursive representation

$$R[x_1, \ldots, x_n] = R[x_1, \ldots, x_{n-1}][x_n].$$

This way one can define $\deg_{x_i}(f)$ for $f \in R[x_1, \ldots, x_n]$ as the degree of f in the polynomial ring $R[x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n][x_i]$, and analogously the leading coefficient as $\operatorname{lc}_{x_i}(f)$.

Definition 2.2. A ring R is called a unique factorization domain (UFD) if it is a domain, that is a ring, where only the zero-element is a zero-divisor,

and any non-zero non-unit element r can be decomposed in the following way: $r = p_1 \cdots p_s$, where p_i is irreducible, and this decomposition is unique up to multiplication by units and reordering.

Proposition 2.1. *If R is a unique factorization domain, then so is $R[x]$.*

Proof: see [Bos06, 2.7 Satz 7]

Thus, any polynomial ring $\mathbb{K}[x]$ over a field $\mathbb{K}$ is a unique factorization domain, and a multivariate polynomial ring $R[\mathbf{x}]$ is a UFD if and only if R is a UFD.

Definition 2.3. Let R be a UFD and $f = \sum_{i=0}^{n} a_i x^i \in R[x]$. The content $\mathrm{cont}(f)$ of f is defined as $\gcd(a_0, \ldots, a_n)$. If $\mathrm{cont}(f) = 1$, f is called primitive. The primitive part $\mathrm{pp}(f)$ of f is defined as $f/\mathrm{cont}(f)$.

Definition 2.4. Let $R[\mathbf{x}]$ be a unique factorization domain. Any non-constant polynomial $f \in R[\mathbf{x}]$ can be decomposed as $f = f_1 \cdots f_r$, where f_i is irreducible. Collecting those factors that are equal, yields $f = f_1^{e_1} \cdots f_s^{e_s}$, where f_i is irreducible, $f_i \neq f_j$ for all $i \neq j$, and $e_i \in \mathbb{N}_{>0}$. If f is a univariate polynomial over a field, we require the factors to be monic (that is, their leading coefficient equals 1) and add $\mathrm{lc}(f)$ as f_1. If f is a univariate polynomial over a UFD, which is not a field, we require the factors to be primitive and add $\mathrm{cont}(f)$ as f_1. We denote this decomposition by $\mathrm{Irr}(f)$.

Enforcing the normalization of the factors in the univariate case is not mandatory, although it is usually done by any implementation that computes $\mathrm{Irr}(f)$. If f is not univariate, it is still possible to normalize the factors. However, we do not require it to keep our following presentation lean.

Since we are concerned with computing the irreducible decomposition of a polynomial, we assume from now on that all rings considered are UFDs.

Let p be a prime number. Then we write $\mathbb{F}_p$ for $\mathbb{Z}/p\mathbb{Z}$ which is a finite field of characteristic p with exactly p elements. By $\mathbb{F}_q$ we denote a finite field with $q = p^n$ elements for some $n \in \mathbb{N}_{>0}$.

For an extensive introduction to finite fields and various ways to represent them, we refer the reader to [LN02].

We now introduce the $\mathcal{O}$ notation, and give some cost estimates for basic univariate operations.

Definition 2.5. $\mathcal{O}$-notation (big-O)

1. A partial function $f : \mathbb{N} \longrightarrow \mathbb{R}$, that is, one that need not be defined for all $n \in \mathbb{N}$, is called eventually positive if there is a constant $N \in \mathbb{N}$ such that $f(n)$ is defined and strictly positive for all $n \geq N$.

2. Let $g : \mathbb{N} \longrightarrow \mathbb{R}$ be eventually positive. Then $\mathcal{O}(g)$ is the set of all eventually positive functions $f : \mathbb{N} \longrightarrow \mathbb{R}$ for which there exist $N, c \in \mathbb{N}$ such that $f(n)$ and $g(n)$ are defined and $f(n) \leq cg(n)$ for all $n \geq N$.

We state some properties of the $\mathcal{O}$-notation:
Let $f, g : \mathbb{N} \longrightarrow \mathbb{R}$ be eventually positive, then:

- $c \cdot \mathcal{O}(f) = \mathcal{O}(f)$ for any $c \in \mathbb{R}_{>0}$
- $\mathcal{O}(f) + \mathcal{O}(g) = \mathcal{O}(f + g) = \mathcal{O}(\max(f, g))$, where one takes the point-wise maximum
- $\mathcal{O}(f) \cdot \mathcal{O}(g) = \mathcal{O}(f \cdot g) = f \cdot \mathcal{O}(g)$
- $f(n) \in g(n)^{\mathcal{O}(1)} \Leftrightarrow f$ is bounded by a polynomial in g.

By $M(n)$ we denote the cost of multiplying two degree n univariate polynomials and assume that M satisfies the following properties for all $m, n \in \mathbb{N}_{>0}$:

- $M(n)/n \geq M(m)/m$ if $n \geq m$
- $M(mn) \leq m^2 M(n)$
- $M(mn) \geq m \cdot M(n)$
- $M(n + m) \geq M(n) + M(m)$
- $M(n) \geq n$

The following table gives an overview of values for M:

Algorithm	$M(n)$
classical	$2n^2$
Karatsuba	$\mathcal{O}(n^{1.59})$
Fast-Fourier Transform (FFT) if R supports FFT	$\mathcal{O}(n \log(n))$
Schönhage & Strassen	$\mathcal{O}(n \log(n) \log\log(n))$

For a detailed description of these algorithms see [vzGG03, Chapter 8].

We assume that

$$M(n) \in \mathcal{O}(n \log(n) \log\log(n))$$

if not mentioned otherwise. Next, we give an overview of the cost of some basic univariate polynomial operations:

operation	cost
division with remainder modular multiplication	$\mathcal{O}(M(n))$
powering with exponent d of modulars	$\mathcal{O}(M(n) \log(d))$
gcd multipoint evaluation reduction modulo several moduli interpolation	$\mathcal{O}(M(n) \log(n))$

Chapter 3

Square-free factorization

To remove multiple factors from a polynomial, one can compute its square-free decomposition. Removing multiple factors from a polynomial is not a mandatory step of the Extended Zassenhaus algorithm and may be omitted.

Definition 3.1. Let R be a ring and $f \in R[\mathbf{x}]$. Then f is called square-free if its decomposition into irreducible polynomials has no repeated non-constant factors. A square-free decomposition of a primitive polynomial f is defined to be $\mathrm{Sqr}(f) = \{(g_1, m_1), \ldots, (g_r, m_r)\}$, $(g_i, m_i) \in (R[\mathbf{x}] \backslash R) \times \mathbb{N}$ for all i, satisfying:

1. $f = \prod_{(g_i, m_i) \in \mathrm{Sqr}(f)} {g_i}^{m_i}$
2. all of the g_i's are pairwise co-prime
3. all of the g_i's are square-free
4. all of the m_i's are distinct

The square-free decomposition is unique up to multiplication by units.

From now on we write f' for the derivative of a univariate polynomial f.

Lemma 3.1. *Let R be a finite field or a field of characteristic zero. Then a non-constant $f \in R[x]$ is square-free if and only if* $\gcd(f, f') = 1$.

Proof: See [vzGG03, Corollary 14.25].

In characteristic $p > 0$ the derivative of a univariate polynomial f may vanish. This is the case if and only if f is a p-th root. In the multivariate

setting, f is a p-th root if and only if $\frac{\partial f}{\partial x_i} = 0$ for all i. Hence, one needs to be able to compute a p-th root of a polynomial to compute the square-free decomposition of a polynomial in positive characteristic. This is, for example, the case if R is a perfect field.

We present an algorithm by Yun [Yun76] for computing the square-free decomposition of a univariate polynomial in characteristic zero, which generalizes to the multivariate case. A modified version serves as a subroutine when computing the square-free decomposition of a multivariate polynomial in positive characteristic.

Algorithm 3.1 (Yun's square-free factorization algorithm)**.**
Input: $f \in R[x]$ monic, where R is a ring of characteristic 0

Output: $\mathrm{Sqr}(f) \subset (R[x] \backslash R) \times \mathbb{N}$ such that $f = \prod_{(g_i, m_i) \in \mathrm{Sqr}(f)} {g_i}^{m_i}$

Instructions:

1. $u = \gcd(f, f')$, $v_1 = \frac{f}{u}$, $w_1 = \frac{f'}{u}$
2. $i = 1$
   while $\deg(v_i) > 0$
   (a) $h_i = \gcd(v_i, w_i - v_i')$, $v_{i+1} = \frac{v_i}{h_i}$, $w_{i+1} = \frac{w_i - v_i'}{h_i}$
   (b) **if** $\deg(h_i) > 0$ **then** append (h_i, i) to $\mathrm{Sqr}(f)$
   (c) $i = i + 1$
3. **return** $\mathrm{Sqr}(f)$

Theorem 3.1. *Let f be a polynomial of degree n. Then the algorithm uses $\mathcal{O}(M(n)\log(n))$ operations in R and it correctly computes the square-free decomposition of f.*

Proof: see [vzGG03, Theorem 14.23]

[vzGG03, Exercise 14.30] modifies Yun's algorithm so as to work over a finite field. For a recent survey and further readings on square-free factorization and the related separable factorization we refer to [Lec08].

For the multivariate case we use an algorithm by L. Bernadin [Ber99]. As a first step, we need a modification of Yun's algorithm:

Algorithm 3.2 (Modified Yun's square-free factorization algorithm)**.**
Input: $f \in \mathbb{F}_{q=p^k}[x_1, \ldots, x_n]$ primitive with respect to each x_i and $x \in \{x_1, \ldots, x_n\}$

Output: as specified in theorem 3.2

Instructions:

1. $u = \gcd(f, \frac{\partial f}{\partial x}), \quad v_1 = \frac{f}{u}, \quad w_1 = \frac{\frac{\partial f}{\partial x}}{u}$
2. $i = 1$
   while $\deg(v_i) > 0$ and $i < p - 1$
   (a) $h_i = \gcd(v_i, w_i - \frac{\partial v_i}{\partial x}), \quad v_{i+1} = \frac{v_i}{h_i}, \quad w_{i+1} = \frac{w_i - \frac{\partial v_i}{\partial x}}{h_i}$
   (b) $u = \frac{u}{v_{i+1}}$
   (c) $i = i + 1$
3. $h_i = v_i$
4. **return** u and $h_1, \ldots, h_i$

Theorem 3.2. *Let $f \in \mathbb{F}_{q=p^k}[x_1, \ldots, x_n]$ be primitive and $x \in x_1, \ldots, x_n$ such that $\frac{\partial f}{\partial x} \neq 0$. Moreover, let*

$$f = \prod_{j=1}^{r} g_j^j = \prod_{(g_i, m_i) \in Sqrf(f)} g_i^{m_i},$$

where $r = \max\{m_i\}$ and some of the g_j may be one. Then the output of the previous algorithm satisfies the following:

1. $$u = \frac{f}{\prod_{i=0}^{p-1} h_i^i} \text{ and } h_i = \prod_{\substack{j=i \\ p|(j-i)}}^{r} k_j,$$

   where k_j is the product of those irreducible factors of g_j, whose derivative with respect to x is not zero

2. $\frac{\partial u}{\partial x} = 0$

Proof: see [Ber99].

Algorithm 3.3 (Square-free factorization)**.**
Input: $f \in \mathbb{F}_{q=p^k}[x_1, \ldots, x_n]$ primitive with respect to each x_i and $x \in \{x_1, \ldots, x_n\}$

Output: $\mathrm{Sqr}(f) \subset (\mathbb{F}_q[\mathbf{x}] \backslash \mathbb{F}_q) \times \mathbb{N}$ such that $f = \prod_{(g_i, m_i) \in \mathrm{Sqr}(f)} {g_i}^{m_i}$

Instructions:

1. **for** all $x_j \in \{x_1, \ldots, x_n\}$ such that $\frac{\partial f}{\partial x_j} \neq 0$
   (a) **call** Algorithm 3.2 with input f and x_j to compute u and $h_i^{(x_j)}$
   (b) $f = u$
2. $h_i = \prod_{x_j \in \{x_1, \ldots, x_n\}} h_i^{(x_j)}$
3. $b = \sqrt[p]{f}$
4. recursively **call** the algorithm with input b to compute $\mathrm{Sqr}(b)$
5. (a) **for** all $(k_l, n_l) \in \mathrm{Sqr}(b)$
      i. **for** all h_i with $\deg(h_i) > 0$ and $0 < i < p$
         if $\deg(\gcd(k_l, h_i)) > 0$ **then**
         $\mathrm{Sqr}(f) = \mathrm{Sqr}(f) \cup \{(g_j, m_j) = (\gcd(k_l, h_i), n_l p + i)\}$
      ii. $k_l = k_l / \prod_{\substack{m_j = n_l p + i \\ 0 < i < p}} g_j$
      iii. **if** $\deg(k_l) > 0$ **then** $\mathrm{Sqr}(f) = \mathrm{Sqr}(f) \cup \{(k_l, n_l p)\}$
   (b) **for** all h_i with $\deg(h_i) > 0$ and $0 < i < p$
      i. $h_i = h_i / \prod_{\substack{m_j = n_l p + i \\ (k_l, n_l) \in \mathrm{Sqr}(b)}} g_j$
      ii. **if** $\deg(h_i) > 0$ **then** $\mathrm{Sqr}(f) = \mathrm{Sqr}(f) \cup \{(h_i, i)\}$
6. **return** $\mathrm{Sqr}(f)$

Proof: see [Ber99].

In the above algorithms we assumed the input to be primitive with respect to each variable. The discussion in [Zip93, Chapter 21.1] suggests that this approach avoids intermediate expression swell when computing the square-free decomposition. Furthermore, one already obtains a coarse decomposition by extracting contents.

As the square-free factorization algorithm is based on repeated GCD computations, the author has also implemented several algorithms to compute multivariate polynomial GCDs: EZGCD [GCL92, Algorithm 7.3], modular GCD [GCL92, Algorithm 7.1, Algorithm 7.2], sparse modular GCD [Zip79],[Zip93],[dKMW05],[Yan09], and a modular GCD over number fields [Enc95].

Chapter 4

Reduction

In the multivariate Extended Zassenhaus factorization algorithm the problem of factoring a multivariate polynomial is reduced to factoring a polynomial in fewer, say i, variables by plugging in a suitable point $a_{i+1}, \ldots, a_n$ for $x_{i+1}, \ldots, x_n$. From a quick glance one is tempted to assume that univariate and multivariate factors are in one-to-one correspondence. However, this does not need to be the case as the following example shows:

Example: Consider $f = 2x^2y - x^2 + 6xy - 44y + 1$ over $\mathbb{Q}$, which is irreducible. Now $f(x, 0) = -x^2 + 1 = -1(x - 1)(x + 1)$. However, $f(x, 1) = x^2 + 6x - 43$ is irreducible.

In fact, this kind of behavior is typical when factoring polynomials over $\mathbb{Q}$.

Over a finite field there are even bivariate polynomials that do never admit a one-to-one correspondence of univariate and bivariate factors, no matter what evaluation point is chosen.

Example: Consider the following irreducible polynomial over $\mathbb{F}_{17}$:

$$f = y^8 + 5x^2y^6 + 10y^6 + 13x^4y^4 + x^2y^4 + 9y^4 + 6x^6y^2 + 2x^4y^2 + 3x^2y^2 + 12y^2 + 9x^8 + 4x^6 + 5x^4 + 5x^2 + 13.$$

$f(a, y)$ splits into 4 factors of degree 2 for every $a \in \mathbb{F}_{17}$.

Let $f \in R[x_1, \ldots, x_n]$ be an irreducible polynomial. A point $a_{i+1}, \ldots, a_n$ such that $f(x_1, \ldots, x_i, a_{i+1}, \ldots, a_n)$ remains irreducible is called *Hilbertian point* for f.

Now if $R = \mathbb{Q}$ or $\mathbb{Q}(\alpha)$, the set of Hilbertian points for f is dense. If R is a finite field, one needs different substitutions, namely, linear polynomials in a new variable to maintain irreducibility with high probability. We give two results:

Theorem 4.1. *[Zip93] Let $f \in \mathbb{Q}[x_1, \ldots, x_n, y]$ be irreducible and let $R(N)$ denote the number of n-tuples $a_1, \ldots, a_n$ over $\mathbb{Z}$ with $|a_i| < N$ such that $f(a_1, \ldots, a_n, y)$ remains irreducible. Then*

$$R(N) < cN^{n-\frac{1}{2}} \log(N),$$

where c solely depends on the degree of f.

Theorem 4.2. *[Kal85a] Let $R = \mathbb{K}[x_1, \ldots, x_n]$ for a perfect field $\mathbb{K}$ and $f \in R[y]$. Moreover, let $d = \deg_y(f)$ and the total degree of f in $x_1, \ldots, x_n$ be D. Then*

$$Prob(f(a_1 + b_1 t, \ldots, a_n + b_n t, y) \text{ is irreducible over } \mathbb{K}[t, y] | a_i, b_i \in S)$$

$$\leq 1 - \frac{4dD2^d}{B}$$

where S is a subset of $\mathbb{K}$ of size B.

From a theoretical point of view, the latter result can be used to design algorithms that reduce multivariate polynomial factorization to bivariate factorization in polynomial time, see [vzGK85a] and [Kal89].
However, from a practical point of view, substituting linear polynomials leads to very large bivariate polynomials that have to be factorized, and thus, to slow implementations.

Hence, similar to [Ber99] we allow evaluation points that may lead to extraneous factors which need to be recombined to obtain true factors. In a naive algorithm one checks all possible combinations by brute force, which means, one has to check 2^{r-1} combinations in the worst-case, where r is the number of modular factors. In practice however, there is almost always a one-to-one correspondence between bivariate and multivariate factors when reducing from a multivariate problem to a bivariate one by plugging in a point $a = (a_3, \ldots, a_n) \in R^{n-2}$. Therefore, we suggest the following reduction strategy:

First, reduce the multivariate problem to a bivariate one; then reduce the bivariate problem to a univariate one.

This strategy also has advantages when precomputing leading coefficients as for this a bivariate factorization is necessary (see Section 6.4). To the best of our knowledge, it is still an open problem to find an effective Hilbert Theorem for these kind of substitutions.

The evaluation points need to satisfy the following conditions:
For a square-free $f \in R[x_1, \ldots, x_n]$ choose a point $a = (a_3, \ldots, a_n) \in R^{n-2}$ such that

1. $g = f(x_1, x_2, a_3, \ldots, a_n)$ remains square-free
2. $\deg_{x_1}(f) = \deg_{x_1}(g)$ and $\deg_{x_2}(f) = \deg_{x_2}(g)$

Now to reduce g, one chooses a_2 in R such that the above holds true for g instead of f.

The first condition is necessary to apply Hensel lifting, and the second one is necessary since by Hensel lifting one cannot reconstruct terms involving x_1 and x_2 of degree higher than $\deg_{x_1}(g)$ respectively $\deg_{x_2}(g)$.

Various different evaluation points may be chosen to obtain a factorization in less variables which has few factors. If one of the reduced polynomials turns out to be irreducible, then by the second condition the original polynomial f is irreducible, too.

For the bivariate problem of computing an irreducible decomposition of g over a finite field, extraneous factors occur frequently. Nevertheless, it is shown in [GL02] that the brute force search for true factors, after lifting to high enough precision, is polynomial-time on average; and there are also algorithms that are polynomial-time in the worst-case, which we discuss later. If $R = \mathbb{Q}$ or $R = \mathbb{Q}(\alpha)$, extraneous factors are rare as the above theorem indicates.

If $\mathbb{F}_q$ or $\mathbb{F}_q^{n-2}$ do not contain any feasible evaluation point, one has to pass to a field extension. As a matter of fact, if one chooses an extension of degree m, where m is prime and larger than the total degree d of the polynomial considered, then the factors over $\mathbb{F}_q$ and $\mathbb{F}_{q^m}$ coincide [vzG85].

Chapter 5

Univariate factorization

Univariate factorization can be seen as a black box in the Zassenhaus approach. So, we only give a quick overview of basic algorithms and some references for the gentle reader. For a detailed description of the basic algorithms we refer the reader to [vzGG03, Chapter 14-16].

Over a finite field classical algorithms due to Berlekamp and Cantor and Zassenhaus can be used. The latter has several derivations, for example, von zur Gathen-Shoup [vzGS92], Kaltofen-Shoup [KS98] and Umans [KU11],[Uma08]. Special variants for polynomials over $\mathbb{F}_2$ are presented in von zur Gathen-Gerhard [vzGG02] and Brent-Zimmermann [BZ07]. Kaltofen and Lobo [KL94] developed a variant of the Berlekamp algorithm, which is based on black-box linear algebra and was later refined in [KS98]. The Kaltofen-Shoup variant of the Cantor-Zassenhaus algorithm is implemented in `NTL` [Sho]. Some techniques used for the implementation of the latter algorithm are explained in [Sho95].
Another possibility to factorize univariate polynomials over a finite field is Niederreiter's algorithm, which is based on solutions of $h^{(p-1)} + h^p = 0$, where $h \in \mathbb{F}_p(x)$ and $h^{(p-1)}$ denotes the $p-1$-th derivative of h, and using linear algebra [Nie93]. For a generalization of this method to arbitrary finite fields see, for example, [FR96].

Over $\mathbb{Z}$ techniques similar to the ones presented here can be used. A basic description of the Berlekamp-Zassenhaus algorithm can be found in [vzGG03]. The first polynomial-time algorithm is the famous LLL-algorithm [LLL82]. Even though being polynomial-time in theory, on most examples the LLL-algorithm is slow compared to the naive algorithm. In fact, using clever conditions on the true factors, one can make the naive algorithm factorize polynomials with an abundance of modular factors

[ASZ00]. A first fast recombination algorithm based on lattice reduction is described in [vH02], later improved and shown to be polynomial-time in [BvHKS09], and further improved in [HvHN11]. The first one of these fast factorization algorithms using lattice reduction is implemented in `NTL`.

Over $\mathbb{Q}(\alpha)$ the best algorithm is due to Belabas [Bel04], based on lattice reduction as well.

Due to lack of time and efficient prerequisite algorithms in `factory` - such as LLL - we have not implemented fast univariate factorization over $\mathbb{Q}(\alpha)$, but Trager's classical norm based algorithm [Tra76]. To get an efficient algorithm, it is necessary to compute the norm, that is resultants, efficiently. As resultants tend to become dense with large coefficients, one can apply a modular resultant algorithm by Collins [Col71]. Another interesting fact is that the resultant based algorithm produces hard-to-factor polynomials over $\mathbb{Z}$ [Enc97]. Therefore, also fast factorization over $\mathbb{Z}$ is necessary.

Chapter 6

Hensel lifting

As we have seen, the Extended Zassenhaus approach requires the ability of recovering the solution in the original ring. This reconstruction can be accomplished by using Hensel lifting which is based on Hensel's Lemma:

Theorem 6.1 (Hensel's Lemma)**.**
Let $m \in R$ be such that $f \equiv g \cdot h \ mod\ m$ and $sg + th \equiv 1$ for f, g, h, s, t in $R[x]$ and $lc(f)$ is not a zero-divisor in R/m. Furthermore, let h be monic, $\deg(f) = \deg(g) + \deg(h)$, $\deg(s) < \deg(h)$, *and* $\deg(t) < \deg(g)$. *Then there exist polynomials g', h', s', t' such that:*

$$f \equiv g'h' \ mod\ m^2$$

and

$$s'g' + t'h' \equiv 1 \ mod\ m^2,$$

h' is monic, $g' \equiv g \ mod\ m$, $h' \equiv h \ mod\ m$, $s' \equiv s \ mod\ m$, $t' \equiv t \ mod\ m$, $\deg(g') = \deg(g)$, $\deg(h') = \deg(h)$, $\deg(s') < \deg(h')$, *and* $\deg(t) < \deg(g)$.

Proof: see [vzGG03, Alg. 15.10]

One can think of the input s and t as the result of the Extended Euclidean algorithm for co-prime g and h. Theorem 6.1 gives a way to lift two factors with quadratic convergence. Note that the assumption of h to be monic is necessary since otherwise any leading coefficient can be imposed on h as the next example shows:

Example: Let $f = gh$ over $\mathbb{F}_5$ with $g = x^3y^2 + 3x^3 + 4x + 1$ and $h = x^2y + 2x^2 + 2x + 2$. Then $\mathrm{lc}_x(g) = y^2 + 3$ and $\mathrm{lc}_x(h) = y + 2$. Modulo y a possible factorization reads $f = (3x^3 + 4x + 1) \cdot (2x^2 + 2x + 2)$. Then $h' = 2x^2 + 4xy + 2x + 4y + 2$ and $g' = 4x^3y + 3x^3 + 2xy + 4x + 3y + 1 \bmod y^2$.

As you can see, $\mathrm{lc}_x(h') \neq \mathrm{lc}_x(h)$, even though $h' \equiv h \bmod y$ and $g' \equiv g \bmod y$ and $f \equiv gh \bmod y^2$. This is due to the introduction of zero-divisors.

However, if one knows $\mathrm{lc}_x(g)$ and $\mathrm{lc}_x(h)$ before the lifting and replaces the leading coefficient accordingly, then the above process yields the correct factors because the leading coefficients of the factors are not zero-divisors. Also note that in the above example the number of terms of g' and h' is larger than in the original g and h. This behavior is typical and can be avoided if the correct leading coefficients are imposed on the factors.

6.1 Bivariate Lifting

We follow the approach in [Ber99] and use linear Hensel lifting to reconstruct the factorization of f. It lifts all factors at once, instead of building a tree and using two factor Hensel lifting as described in [vzGG99]. Even though, the algorithm from [vzGG99] has quadratic convergence, we found it less performant than the linear algorithm since it involves division of multivariate polynomials. However, it might be advantageous to use this algorithm or a modification of the below algorithm with quadratic convergence, also described in [Ber99], if the lift bound exceeds a certain threshold, say 1000 or more. Another way to lift with quadratic convergence is given in [BLS+04], based on fast computation of sub-product trees and applying Telegen's principle.

Let $f^{(k)}$ denote $f \bmod y^{k+1}$ and $f = \prod_{i=1}^{r} f_i^{(0)}$ be the univariate factorization of $f \bmod y$, where all factors are monic. If f has $\mathrm{lc}_x(f)$ different from 1, it can be added as an additional factor.

Now to lift a factorization from y^k to y^{k+1}, the following Diophantine equation has to be solved:

$$f - \prod_{i=1}^{r} f_i^{(k-1)} \equiv \sum_{i=1}^{r} \delta_i(k) y^k \prod_{j \neq i} f_j^{(k-1)} \bmod y^{k+1} \tag{6.1.1}$$

such that $\deg(\delta_i(k)) < \deg\left(f_i^{(0)}\right)$.

Then $f_i^{(k)} = \delta_i(k) y^k + f_i^{(k-1)}$ satisfies $f - \prod_{i=1}^{r} f_i^{(k)} \equiv 0 \bmod y^{k+1}$.

Since $f - \prod_{i=1}^{r} f_i^{(k-1)} \equiv 0 \bmod y^k$, the Diophantine equation above reads:

$$\frac{f - \prod_{i=1}^{r} f_i^{(k-1)}}{y^k} \equiv \sum_{i=1}^{r} \delta_i(k) \prod_{j \neq i} f_j^{(0)} \bmod y.$$

Now let $e = f - \prod_{i=1}^{r} f_i^{(k-1)}$ denote the error and $e^{[k]}$ the coefficient of y^k in e. Let δ_i be the solution of the univariate Diophantine equation

$$1 \equiv \sum_{i=1}^{r} \delta_i \prod_{j \neq i} f_j^{(0)} \bmod y. \tag{6.1.2}$$

Then $\delta_i(k) = e^{[k]}\delta_i \bmod f_i^{(0)}$ is a solution of (6.1.1).

[Ber99] gives a fast way to compute the product $\prod_{i=1}^{r} f_i^{(k-1)}$, and hence the error e.
We define $U_j^{(k)} = \prod_{i=1}^{j} f_i^{(k-1)}$. Then $U_2^{(k)}$ reads as follows:

$$\begin{aligned}
&f_1^{[0]} f_2^{[0]} + \\
&\left(f_1^{[1]} f_2^{[0]} + f_1^{[0]} f_2^{[1]}\right) y + \\
&\left(f_1^{[2]} f_2^{[0]} + f_1^{[1]} f_2^{[1]} + f_1^{[0]} f_2^{[2]}\right) y^2 + \\
&\left(f_1^{[3]} f_2^{[0]} + f_1^{[2]} f_2^{[1]} + f_1^{[1]} f_2^{[2]} + f_1^{[0]} f_2^{[3]}\right) y^3 + \\
&\left(f_1^{[4]} f_2^{[0]} + f_1^{[3]} f_2^{[1]} + f_1^{[2]} f_2^{[2]} + f_1^{[1]} f_2^{[3]} + f_1^{[0]} f_2^{[4]}\right) y^4 + \\
&\vdots
\end{aligned}$$

And $U_j^{(k)}$ as:

$$\begin{aligned}
&U_{j-1}^{[0]} f_j^{[0]} + \\
&\left(U_{j-1}^{[1]} f_j^{[0]} + U_{j-1}^{[0]} f_j^{[1]}\right) y + \\
&\left(U_{j-1}^{[2]} f_j^{[0]} + U_{j-1}^{[1]} f_j^{[1]} + U_{j-1}^{[0]} f_j^{[2]}\right) y^2 + \\
&\left(U_{j-1}^{[3]} f_j^{[0]} + U_{j-1}^{[2]} f_j^{[1]} + U_{j-1}^{[1]} f_j^{[2]} + U_{j-1}^{[0]} f_j^{[3]}\right) y^3 + \\
&\left(U_{j-1}^{[4]} f_j^{[0]} + U_{j-1}^{[3]} f_j^{[1]} + U_{j-1}^{[2]} f_j^{[2]} + U_{j-1}^{[1]} f_j^{[3]} + U_{j-1}^{[0]} f_j^{[4]}\right) y^4 + \\
&\vdots
\end{aligned}$$

Now the $U_j^{(k)}$ can be computed even faster than in [Ber99] by using Karatsuba's trick:
First, compute $M_j^{[0]} = U_{j-1}^{[0]} f_j^{[0]}$ and $M_j^{[1]} = U_{j-1}^{[1]} f_j^{[1]}$, then $U_j^{[1]}$ can be computed as $\left(U_{j-1}^{[0]} + U_{j-1}^{[1]}\right)\left(f_j^{[0]} + f_j^{[1]}\right) - M_j^{[0]} - M_j^{[1]}$. The $M_j^{[k]}$ can be stored and then the computation of $U_j^{[2]}$ only takes two multiplications in contrast to the original three.

Another advantage of Bernardin's Hensel lifting is that one needs to compute δ_i only once, and $\delta_i(k)$ can be easily computed by one multiplication and one division with remainder. In the tree-Hensel lifting algorithm one needs to lift not only the factors themselves but also the corresponding Bézout coefficients.

Algorithm 6.1 (Bivariate Hensel lifting)**.**
Input: $f \in R[x][y]$, $f \equiv \prod_{i=1}^{r} f_i^{(0)} \bmod y$, where the $f_i^{(0)}$ are pairwise co-prime, $\mathrm{lc}_x(f) = f_1^{(0)}$ not vanishing mod y, $f_i^{(0)}$ is monic in x for $i = 2, \ldots, r$, and an integer $l \geq 1$

Output: $f_i^{(l-1)} \in \mathrm{Quot}(R)[x][y]$ such that $f \equiv \prod_{i=1}^{r} f_i^{(l-1)} \bmod y^l$ and $f_i^{(l-1)} \equiv f_i^{(0)} \bmod y$ for all $i = 2, \ldots, r$

Instructions:

1. initialize $U_2^{[0]} = f_1^{[0]} f_2^{[0]}, \ldots, U_r^{[0]} = U_{r-1}^{[0]} f_r^{[0]}$, and $M_i^{[0]} = U_i^{[0]}$ **for** $i = 2, \ldots, r$.
2. solve $1 \equiv \sum_{i=1}^{r} \delta_i \prod_{j \neq i} f_j^{(0)}$
3. **for** $k = 1, \ldots, l-1$
   call the next algorithm 6.2 with input $f, f_i^{(k-1)}, f_i^{(0)}, \delta_i, U_i, M_i,$ and k
4. **return** $f_i^{(l-1)}$

For convenience, we set $U_1 = f_1$ in the following.

Algorithm 6.2 (Hensel step)**.**
Input: as in algorithm 6.1 with additional input $f_i^{(k-1)}, \delta_i, U_i, M_i$, and an integer $k \geq 1$

Output: $f_i^{(k)} \in \mathrm{Quot}(R)[x][y]$ such that $f \equiv \prod_{i=1}^{r} f_i^{(k)} \bmod y^{k+1}$ and $f_i^{(k)} \equiv f_i \bmod y$ for all $i = 2, \ldots, r$, M_i such that $M_i^{(k)} = U_{i-1}^{(k)} f_i^{(k)}$, and $U_i = \prod_{j=1}^{i} f_j^{(k)} \bmod y^{k+2}$ for all $i = 2, \ldots, r$.

Instructions:

1. $e = f^{[k]} - U_r^{[k]}$
2. $\delta_i(k) = \delta_i \cdot e \bmod f_i^{(0)}$ **for** $i = 1, \ldots, r$
3. $f_i^{[k]} = \delta_i(k)$ **for** $i = 1, \ldots, r$
4. **for** $j = 2, \ldots, r$
   (a) $t_j = 0$
   (b) $M_j^{[k]} = U_{j-1}^{[k]} f_j^{[k]}$
   (c) $U_j^{[k]} = U_j^{[k]} + \left(U_{j-1}^{[0]} + U_{j-1}^{[k]}\right)\left(f_j^{[0]} + f_j^{[k]}\right) - M_j^{[k]} - M_j^{[0]}$
   (d) **for** $m = 1, \ldots, \lceil k/2 \rceil$
      i. **if** $m \neq k - m + 1$
      $t_j =$
      $t_j + \left(U_{j-1}^{[0]} + U_{j-1}^{[k-m+1]}\right)\left(f_j^{[0]} + f_j^{[k-m+1]}\right) - M_j^{[k]} - M_j^{[k-m+1]}$
      ii. **else**
      $t_j = t_j + M_j^{[k]}$
   (e) $U_j^{[k+1]} = t_j$
5. **return** $f_i^{(k)}, U_i, M_i$

Now one can use the following algorithm to compute δ_i in step 2 of algorithm 6.1:

Algorithm 6.3 (univariate Diophantine equation)**.**
Input: pairwise co-prime polynomials $f_1, \ldots, f_r \in \mathrm{Quot}(R)[x]$

Output: δ_i such that $1 = \sum_{i=1}^{r} \delta_i \prod_{j \neq i} f_j$ and $\deg(\delta_i) < \deg(f_i)$

Instructions:

1. compute $p_i = \prod_{j \neq i} f_j$
2. compute $h_2 = \gcd(p_1, p_2) = s_1 p_1 + s_2 p_2$
3. **for** $i = 3, \ldots, r$
   (a) $h_i = \gcd(h_{i-1}, p_i) = s_{i+1} h_{i-1} + s_i p_i$
   (b) **for** $j = 1, \ldots, i - 1$

i. $s_j = s_j s_{i+1}$
ii. $s_j = s_j \bmod f_j$

4. **return** $s_1, \ldots, s_r$

Proof: see [GCL92]

Note that algorithm 6.1 can be halted and resumed at any stage of the lifting if one buffers U_i, M_i, and δ_i.

For the complexity estimate, note that at step k of the lifting one needs to compute $\lceil k/2 \rceil + 2$ multiplications per U_j. Now assuming that each factor has degree m/r in x, this yields

$$\begin{aligned}
&\mathcal{O}\left(\sum_{j=2}^{r} (\lceil k/2 \rceil + 2) M\left(\frac{(j-1)m}{r}\right)\right) \\
&\leq \mathcal{O}\left(\sum_{j=2}^{r} (\lceil k/2 \rceil + 2)(j-1) M\left(\frac{m}{r}\right)\right) \\
&\leq \mathcal{O}\left(\frac{r^2}{2}(\lceil k/2 \rceil + 2) M\left(\frac{m}{r}\right)\right).
\end{aligned}$$

If naive multiplication is used, this gives $\mathcal{O}\left(\frac{k}{4}m^2\right)$ which is to be compared with the original estimate of $\mathcal{O}\left(\frac{k}{2}m^2\right)$ in [Ber99]. Though we did not improve on the asymptotic complexity here, we improved on the constant. Finally to lift up to precision n, the time amounts to $\mathcal{O}(n^2 m^2)$ omitting any constant factor.
For the total running time of Hensel lifting, note that the initial Diophantine equation in step 2 of algorithm 6.1 can be solved in $\mathcal{O}(rM(m)\log(m))$. It takes $\mathcal{O}(rM(m))$ to solve for the $\delta_i(k)$ at step k. Hence, the overall time is

$$\mathcal{O}\left(r^2 n^2 M\left(\frac{m}{r}\right) + r(nM(m) + \log(m))\right).$$

One may need to pass to the quotient field of R since the factors are required to be monic in x. This can lead to very large coefficients and makes solving the initial Diophantine equation very expensive as one needs to compute the Bézout coefficients. For instance, over $\mathbb{Q}$ the absolute value of the numerator and the denominator of the Bézout coefficients can be

bounded by $(n+1)^n A^{n+m}$ if the input of the Extended Euclidean algorithm is of degree n and m ($n \geq m$) respectively and max-norm less than A [vzGG03]; wheras the bound on the coefficients of a bivariate factor can be much less as shown in the following.

Therefore, we proceed in the following way:
First, a bound B on the coefficients of a divisor of f is computed. Then one chooses a suitable prime number p and uses algorithm 6.6 to lift a solution of 6.1.2 modulo p to a solution modulo p^k, where k is chosen such that $p^k > 2B$. Over a number field $\mathbb{Q}(\alpha)$ with minimal polynomial μ, $(\mathbb{Z}/p\mathbb{Z})[t]/(\mu_p)$ may contain zero-divisors, as the reduction μ_p of μ modulo p may be reducible. Therefore, one may encounter zero-divisors when solving 6.1.2 modulo p. In this case, a different p has to be chosen. By [LM89, Theorem 3.2] there are only finitely many primes for which this fails. If one passes a solution δ_i of 6.1.2 modulo p to algorithm 6.6, then step 4a in 6.6 cannot fail since $f_i^{(0)}$ is monic.
All subsequent steps of algorithms 6.4 and 6.5 are performed modulo p^k. When reducing in step 2 of algorithm 6.2 zero-divisors are never encountered because all factors are monic in x.

The approach of bounding coefficients of a factor and computing modulo some power of a prime is classical in the integer case (see e.g. [Mus75], [WR75]). In the number field case similar techniques are used (see e.g. [Wan76], [JM09]). However, to the best of our knowledge ours is the only one that neither requires α to be an algebraic integer, nor μ to stay irreducible modulo p, nor uses a heuristic bound.

The input specifications of algorithm 6.1 have to be altered accordingly:

Algorithm 6.4 (Bivariate Hensel lifting over $\mathbb{Z}$)**.**
Input: $f \in \mathbb{Z}[x][y]$, $f \equiv \prod_{i=1}^{r} f_i^{(0)} \bmod y$, where the $f_i^{(0)}$ are pairwise co-prime, $\mathrm{lc}_x(f) = f_1$ not vanishing mod y, $f_i^{(0)}$ is primitive for $i = 2, \ldots, r$, an integer $l \geq 1$, B a bound on the coefficients of the factors of f, a prime p such that $p \nmid \mathrm{lc}_x(f)(0)$, $f_i^{(0)}$ are pairwise co-prime mod p, and $k > 0$ such that $p^k > 2B$

Output: $f_i^{(l-1)} \in (\mathbb{Z}/p^k\mathbb{Z})[x][y]$ such that $f \equiv \prod_{i=1}^{r} f_i^{(l-1)} \bmod < y^l, p^k >$ and $f_i^{(l-1)} \equiv \mathrm{lc}\left(f_i^{(0)}\right)^{-1} f_i^{(0)} \bmod < y, p^k >$ for all $i = 2, \ldots, r$

Instructions:

1. **for** $i = 2, \ldots, r$ $f_i^{(0)} = \mathrm{lc}\left(f_i^{(0)}\right)^{-1} f_i^{(0)} \bmod p^k$

2. initialize $U_2^{[0]} = f_1^{[0]} f_2^{[0]}, \ldots, U_r^{[0]} = U_{r-1}^{[0]} f_r^{[0]}$, and $M_i^{[0]} = U_i^{[0]}$ **for** $i = 2, \ldots, r$.

3. solve $1 \equiv \sum_{i=1}^{r} \delta_i^{(p)} \prod_{j \neq i} f_j^{(0)} \bmod p$

4. **call** algorithm 6.6 with input p, k, $f_i^{(0)}$, $\prod_{j \neq i} f_j^{(0)}$, and $\delta_i^{(p)}$ to obtain a solution δ_i of 6.1.2 mod p^k

5. **for** $j = 1, \ldots, l-1$
   call algorithm 6.2 with input $f, f_i^{(j-1)}, f_i^{(0)}, \delta_i, U_i, M_i, p, k,$ and j

6. **return** $f_i^{(l-1)}$

Algorithm 6.5 (Bivariate Hensel lifting over $\mathbb{Q}(\alpha)$)**.**
Input: $f \in \mathbb{Z}[\alpha][x][y]$, $f \equiv \prod_{i=1}^{r} f_i^{(0)} \bmod y$, where the $f_i^{(0)}$ are pairwise co-prime, $\mathrm{lc}_x(f) = f_1$ not vanishing mod y, $f_i^{(0)}$ is primitive for $i = 2, \ldots, r$, an integer $l \geq 1$, B a bound on the coefficients of the factors of f, a prime p such that $p \nmid \mathrm{lc}_x(f)(0)\mathrm{disc}(\mu)$, $f_i^{(0)}$ are pairwise co-prime mod p, and $k > 0$ such that $p^k > 2B$

Output: a prime $\tilde{p}$, $\tilde{k} > 0$ which satisfy the conditions in the input specifications, $f_i^{(l-1)} \in (\mathbb{Z}/\tilde{p}^{\tilde{k}}\mathbb{Z})[t]/(\mu_p)[x][y]$ such that $f \equiv \prod_{i=1}^{r} f_i^{(l-1)} \bmod < y^l, \tilde{p}^{\tilde{k}} >$, and $f_i^{(l-1)} \equiv \mathrm{lc}\left(f_i^{(0)}\right)^{-1} f_i^{(0)} \bmod < y, \tilde{p}^{\tilde{k}} >$ for all $i = 2, \ldots, r$

Instructions:

1. **for** $i = 2, \ldots, r$ $f_i^{(0)} = \mathrm{lc}\left(f_i^{(0)}\right)^{-1} f_i^{(0)} \bmod p^k$

2. solve $1 \equiv \sum_{i=1}^{r} \delta_i^{(p)} \prod_{j \neq i} f_j^{(0)} \bmod p$

3. **if** the previous step failed, choose a different prime $\tilde{p}$ and $\tilde{k}$ which satisfy the same conditions as the input p and k, set $p = \tilde{p}$, $k = \tilde{k}$ and **goto** step 1

4. **else** $\tilde{p} = p$, $\tilde{k} = k$

5. **call** algorithm 6.6 with input $\tilde{p}$, $\tilde{k}$, $f_i^{(0)}$, $\prod_{j \neq i} f_j^{(0)}$, and $\delta_i^{(\tilde{p})}$ to obtain a solution δ_i of 6.1.2 mod $\tilde{p}^{\tilde{k}}$

6. initialize $U_2^{[0]} = f_1^{[0]} f_2^{[0]}, \ldots, U_r^{[0]} = U_{r-1}^{[0]} f_r^{[0]}$, and $M_i^{[0]} = U_i^{[0]}$ **for** $i = 2, \ldots, r$.

7. **for** $j = 1, \ldots, l-1$
   call algorithm 6.2 with input $f, f_i^{(j-1)}, f_i^{(0)}, \delta_i, U_i, M_i, \tilde{p}, \tilde{k},$ and j

8. **return** $f_i^{(l-1)}$, $\tilde{p}$, and $\tilde{k}$

Algorithm 6.6 (Diophantine equation)**.**
Input: a prime p, $k > 0$, $f_1, \ldots, f_r \in (\mathbb{Z}/p^k\mathbb{Z})[x]$ or $(\mathbb{Z}/p^k\mathbb{Z})[t]/(\mu_p)[x]$ monic, such that $f_i \bmod p$ are pairwise co-prime, products $P_i = \prod_{j \neq i} f_j$, and solutions $\delta_i^{(p)}$ of 6.1.2 mod p

Output: δ_i such that $1 - \sum_i^r \delta_i P_i = 0 \bmod p^k$

Instructions:

1. set $\delta_i = \delta_i^{(p)}$ **for** $i = 1, \ldots, r$

2. **if** $k = 1$, **return** δ_i

3. set $e = 1 - \sum_i^r \delta_i P_i \bmod p^2$

4. **for** $j = 2, \ldots, k$

   (a) set $\delta_i^{(p^j)} = e^{[p^{j-1}]} \delta_i^{(p)} \bmod f_i^{[p]}$, where $g^{[p^l]}$ denotes the coefficient of p^l in the p-adic expansion of g

   (b) $\delta_i = \delta_i + p^{j-1} \delta_i^{(p^j)}$ **for** $i = 1, \ldots, r$

   (c) $e = 1 - \sum_{i=1}^r \delta_i P_i \bmod p^{j+1}$

5. **return** $\delta_1, \ldots, \delta_r$

Algorithm 6.6 is an adaption of [GCL92, Algorithm 6.3.].

Now how can we bound the coefficients? A result by Gelfond [Gel03] reads: If f is a polynomial in s variables over the complex numbers, $f = f_1 \cdot \ldots \cdot f_r$, n_k is the degree of f in the k-th variable and $n = n_1 + \ldots + n_s$, then

$$\|f_1\|_\infty \cdot \ldots \cdot \|f_r\|_\infty \leq ((n_1 + 1) \cdot \ldots \cdot (n_s + 1)/2^s)^{1/2} \cdot 2^n \cdot \|f\|_\infty$$

Therefore, over $\mathbb{Z}$ one can choose

$$B = ((n_1 + 1)(n_2 + 1))^{1/2} \cdot 2^{n-1} \cdot \|f\|_\infty \tag{6.1.3}$$

and p needs to satisfy $p \nmid \mathrm{lc}(f) \bmod y$.

Over an algebraic number field the situation is more involved, so we first need to introduce some notation:
The minimal polynomial is denoted by $\mu \in \mathbb{Z}[t]$, D denotes the discriminant of μ, and by d_f we denote $\mathrm{res}(\mu, \mathrm{lc}(f))$. Then by [Enc95, Theorem 3.2] if $f \in \mathbb{Z}[\alpha][\mathbf{x}]$ and g is a monic factor of f over $\mathbb{Q}(\alpha)$, then

$$g \in (1/d_f\, d)\mathbb{Z}[\alpha][\mathbf{x}]$$

where d^2 divides D.

Let k be the degree of μ and l the leading coefficient of μ. Using the norms from [Enc95], for a polynomial

$$h = \sum_i h_i \mathbf{x}^i \in \mathbb{Q}(\alpha)[\mathbf{x}]$$

where $h_i = \sum_{j=0}^{k-1} h_{ij}\alpha^j$, $h_{ij} \in \mathbb{Q}$, we set

$$\|h\|_\infty = \max_{ij}(|h_{ij}|)$$

and

$$\|h\|_2 = \max_j \left(\left(\sum_i |h_i^{(j)}|^2 \right)^{1/2} \right)$$

where the maximum is taken over all k conjugates of h_i. Furthermore, we denote by $\tilde{h}$ the monic associate of h.

Theorem 6.2. *Any monic divisor g of f over $\mathbb{Q}(\alpha)$ satisfies:*

$$\|d_f Dg\|_\infty \leq \left(\prod_{i=1}^{s} (n_i + 1) \right) 2^{n+k-s/2} l^{-k} (k+1)^{7k/2} \|f\|_\infty^k \|\mu\|_\infty^{4k} \qquad (6.1.4)$$

Proof: By Gelfond [Gel03] each coefficient η of g satisfies

$$\|\eta\|_2 \leq ((n_1 + 1) \cdot \ldots \cdot (n_s + 1)/2^s)^{1/2} 2^n \|\tilde{f}\|_2$$

Now by [Enc95, Proof of Lemma 4.1]:

$$\|d_f Dg\|_\infty \leq \left(\left(\prod_{i=1}^{s} (n_i + 1) \right) /2^s \right)^{1/2} 2^n \|\tilde{f}\|_2 |D|^{1/2} |d_f| (k+1)^{3k/2} \|\mu\|_\infty^k$$

From

$$\|\tilde{f}\|_2 \leq ((n_1+1)\cdot\ldots\cdot(n_s+1))^{1/2}\|\tilde{f}\|_\infty \sum_{j=0}^{k-1}\|\alpha\|_2^j$$

and

$$\|\alpha\|_2 \leq 2l^{-1}\|\mu\|_\infty$$

one obtains

$$\|\tilde{f}\|_2 \leq ((n_1+1)\cdot\ldots\cdot(n_s+1))^{1/2}\|\tilde{f}\|_\infty 2^k l^{-k}\|\mu\|_\infty^k$$

By Hadamard's bound, $|D| \leq (k+1)^{2k}\|\mu\|_\infty^{2k}$. Hence,

$$\|d_f Dg\|_\infty \leq \left(\prod_{i=1}^{s}(n_i+1)\right) 2^{n+k-s/2}l^{-k}\|d_f\tilde{f}\|_\infty (k+1)^{5k/2}\|\mu\|_\infty^{3k}$$

By [Enc95, Proof of Lemma 4.1],

$$\|d_f\tilde{f}\|_\infty \leq ((k+1)\|\mu\|_\infty\|f\|_\infty)^k$$

6.2 Multivariate Lifting

Now the modular factors are lifted to bivariate ones. To proceed further, we assume the following: The bivariate factors are in one-to-one correspondence to the multivariate factors and the leading coefficient of each multivariate factor is known.

Let

$$\begin{aligned}
F &= \prod_{i=1}^{r} F_i, \\
f &= F(x_1, x_2, 0, \ldots, 0) = \prod_{i=1}^{r} f_i, \\
f_i &= F_i \bmod (x_3, x_4, \ldots, x_n), \\
\mathrm{lc}_{x_1}(F) &= \prod_{i=1}^{r}\mathrm{lc}_{x_1}(F_i) = \prod_{i=1}^{r} l_i, \\
\deg_{x_1}(F) &= \deg_{x_1}(f) = \deg(f(x_1,0)), \\
f_i(x_1,0) &\text{ are pairwise co-prime}
\end{aligned} \tag{6.2.1}$$

Without loss of generality and to simplify the presentation, we restrict to the case $n = 3$.
As above, we want to lift from x_3^k to x_3^{k+1}. If the f_i's were monic in x_1, one could use the above Ansatz, that is, compute a solution of

$$1 \equiv \sum_{i=1}^{r} \delta_i \prod_{j \neq i} f_j \bmod x_3$$

and obtain $\delta_i(k) = e^{[k]}\delta_i \bmod f_i$. However, this is prohibitive as it involves costly division with remainder of multivariate polynomials. Instead, one solves for $\delta_i(k)$ directly. The equation

$$e^{[k]} = \sum_{i=1}^{r} \delta_i(k) \prod_{i \neq j} f_j$$

can be solved by solving it modulo x_2 and lifting the solution. Note that $\prod_{i \neq j} f_j$ needs to be computed only once.

Algorithm 6.7 (Diophantine equation)**.**
Input: $f_1, \ldots, f_r \in R[x_1, \ldots, x_n]$, such that $f_i \bmod (x_2, \ldots, x_n)$ are pairwise co-prime, products $p_i = \prod_{j \neq i} f_j$, positive integers $k_2, \ldots, k_n$, and $E \in R[x_1, \ldots, x_n]$

Output: $\delta_i \in \mathrm{Quot}(R)[x_1, \ldots, x_n]$ such that
$E \equiv \sum_{i=1}^{r} \delta_i p_i \bmod \left(x_2^{k_2}, \ldots, x_n^{k_n}\right)$

Instructions:

1. **if** $n = 1$, solve the univariate Diophantine equation $E = \sum_{i=1}^{r} \delta_i p_i$ and **return** $\delta_1, \ldots, \delta_r$
2. set $p_i' = p_i \bmod x_n$, $f_i' = f_i \bmod x_n$, $E' = E \bmod x_n$ **for** $i = 1, \ldots, r$
3. **call** the present algorithm with input $f_1', \ldots, f_r', p_1', \ldots, p_r', k_2, \ldots, k_{n-1}$, and E' to obtain $\delta_1', \ldots, \delta_r'$
4. $\delta_i = \delta_i'$ **for** $i = 1, \ldots, r$
5. $e = E - \sum_{i=1}^{r} \delta_i' p_i \bmod \left(x_2^{k_2}, \ldots, x_{n-1}^{k_{n-1}}\right)$
6. **for** $j = 1, \ldots, k_n - 1$ and $e \neq 0$

   (a) **call** the present algorithm with input $f'_1, \ldots, f'_r, p'_1, \ldots, p'_r, k_2, \ldots, k_{n-1}, e^{[j]}$ to obtain $\delta''_1, \ldots, \delta''_r$
   (b) $\delta_i = \delta_i + x_n^j \delta''_i$ **for** $i = 1, \ldots, r$
   (c) $e = e - \sum_{i=1}^r x_n^j \delta''_i p_i \bmod \left(x_2^{k_2}, \ldots, x_{n-1}^{k_{n-1}}\right)$

7. **return** $\delta_1, \ldots, \delta_r$

The above algorithm is a generalization of algorithm 6.6 to the multivariate case. Its original description can be found in [GCL92, Algorithm 6.2.].

Since the true leading coefficients are known in advance, they can be imposed on each factor before each lifting step.

Algorithm 6.8 (Multivariate Hensel lifting)**.**
Input: as specified in 6.2.1, and positive integers $k_3, \ldots, k_n$

Output: $g_1, \ldots, g_r$ such that $F \equiv \prod_{i=1}^r g_i \bmod \left(x_3^{k_3}, \ldots, x_n^{k_n}\right)$

Instructions:

1. $I = \emptyset$
2. **for** $l = 3, \ldots, n$
   (a) $f' = F \bmod (x_{l+1}, \ldots, x_n)$
   (b) replace $\mathrm{lc}_{x_1}(f_i)$ by $l_i \bmod (x_{l+1}, \ldots, x_n)$ **for** $i = 1, \ldots, r$
   (c) $U_2^{[0]} = f_1^{[0]} f_2^{[0]}, \ldots, U_r^{[0]} = U_{r-1}^{[0]} f_r^{[0]}$, and $M_i^{[0]} = U_i^{[0]}$ **for** $i = 2, \ldots, r$
   (d) $U_2^{[1]} = f_1^{[1]} f_2^{[0]} + f_1^{[0]} f_2^{[1]}, \ldots, U_r^{[1]} = U_{r-1}^{[1]} f_r^{[0]} + U_{r-1}^{[0]} f_r^{[1]}$
   (e) compute $p_i = \prod_{j \neq i} f_j$ **for** $i = 1, \ldots, r$
   (f) set $f_i^{(0)} = f_i$ **for** $i = 1, \ldots, r$
   (g) **for** $m = 1, \ldots, k_l - 1$
      i. **call** the next algorithm 6.9 with input $f', f_i^{(m-1)}, f_i, p_i, U_i, M_i, I,$ and m
   (h) set $f_i = f_i^{(k_l)}$ **for** $i = 1, \ldots, r$
   (i) $I = I \cup \{x_l^{k_l}\}$
3. **return** $f_1, \ldots, f_r$

Algorithm 6.9 (Hensel step)**.**
Input: $f', f_i^{(m-1)}, f_i, p_i, U_i, M_i, I$, and an integer $m \geq 1$

Output: $f_i^{(m)} \in R[x_1, \ldots, x_l]$ such that $f' \equiv \prod_{i=1}^{r} f_i^{(m)} \bmod < I, x_l^{m+1} >$ and $f_i^{(m)} \equiv f_i \bmod x_l$ for all $i = 1, \ldots, r$, M_i such that $M_i^{[m]} = U_{i-1}^{[m]} f_i^{[m]}$,$M_i^{[m+1]} = U_{i-1}^{[m+1]} f_i^{[m+1]}$, and $U_i = \prod_{j=1}^{i} f_j^{(m+1)} \bmod x_l^{m+2}$ for all $i = 2, \ldots, r$.

Instructions:

1. $e = f'^{[m]} - U_r^{[m]}$
2. **call** algorithm 6.7 with input $f_i, p_i, k_2 = \deg_{x_2}(f'), k_3, \ldots, k_{l-1}, m, e$ to obtain $\delta_i^{(m)}$
3. $f_i^{[m]} = f_i^{[m]} + \delta_i^{(m)}$
4. **for** $j = 2, \ldots, r$
   (a) $t_j = 0$
   (b) $M_j^{[m]} = U_{j-1}^{[m]} f_j^{[m]}$, $M_j^{[m+1]} = U_{j-1}^{[m+1]} f_j^{[m+1]}$
   (c) $U_j^{[m]} = U_j^{[m]} + U_{j-1}^{[0]} f_j^{[m]} + U_{j-1}^{[m]} f_j^{[0]}$
   (d) **for** $l = 1, \ldots, \lceil m/2 \rceil$
      i. **if** $l \neq m - l + 1$
      $t_j =$
      $t_j + \left(U_{j-1}^{[0]} + U_{j-1}^{[m-l+1]}\right)\left(f_j^{[0]} + f_j^{[m-l+1]}\right) - M_j^{[m]} - M_j^{[m-l+1]}$
      ii. **else**
      $t_j = t_j + M_j^{[m]}$
   (e) $t_j = t_j + U_{j-1}^{[0]} f_j^{[m+1]} + U_{j-1}^{[m+1]} f_j^{[0]}$
   (f) $U_j^{[m+1]} = t_j$
5. **return** $f_i^{(m)}, U_i,$ and M_i

The algorithm differs from algorithm 6.2 only in the way the δ_i's are computed. Small adjustments have to be made to the way the U_i's are computed due to the a priori knowledge of the leading coefficients. Essentially, algorithm 6.8 coincides with [GCL92, Algorithm 6.4.], though we have adapted it such that it uses the faster error computation of [Ber99].

6.3 Sparse Lifting

As most multivariate polynomials are sparse, one can also apply sparse heuristic methods for Hensel lifting. Zippel has proposed a sparse variant of Hensel lifting in [Zip93] similar to his sparse modular interpolation. However, it constructs a solution only variable by variable, whereas the below approach by Lucks [Luc86] constructs a solution at once.

Let $F = \prod_{i=1}^{r} F_i$ be the irreducible decomposition of F in $R[x_1, x_2, \ldots, x_n]$ and $f = \prod_{i=1}^{r} f_i$ be the decomposition of $F(x_1, x_2, a_3, \ldots, a_n) = f$ in $R[x_1, x_2]$. Furthermore, assume $F_i(x_1, x_2, a_3, \ldots, a_n) = f_i$ for all i and $l_i = \mathrm{lc}_{x_1}(F_i)$ are given.

The aim is to compute F_i from f_i. For that, consider F and F_i as elements of $R[x_3, \ldots, x_n][x_1][x_2]$. Now each coefficient $F^{(\alpha,\beta)}$ of F is of the form

$$\sum_{(\eta_i,\nu_i)} \prod_{\substack{|\eta|=\alpha \\ |\nu|=\beta}} F_i^{(\eta_i,\nu_i)}$$

From the above equation, one can derive a system of equations for each coefficient of F by replacing the coefficient of $f_i^{(\eta_i,\nu_i)}$ by some indeterminant $\Lambda_{\eta_i,\nu_i}^{(i)}$.
In the dense case this system of equation is dense and large either. However, if the bivariate factors are sparse and F is too, only few equations arise. To actually solve this system, one can feed it the precomputed leading coefficients l_i as starting solution, that is, all $\Lambda_{d_i,\nu_i}^{(i)}$, where $d_i = \deg_{x_1}(f_i)$, are known. So, after plugging in these values, one searches the system for linear equations, solves those, and plugs in the newly found solutions, and so on. If there are no linear equations, one can try more elaborate system solving methods or abandon the heuristic. It is also possible to not only precompute the leading coefficient but also the trailing coefficient. This way more starting solutions are known and the heuristic is more likely to succeed.

Algorithm 6.10 (Lucks' heuristic sparse lifting)**.**
Input: $F \in R[x_1, x_2, \ldots, x_n]$, $F(x_1, x_2, a_3, \ldots, a_n) = f = \prod_{i=1}^{r} f_i$, and $\deg_{x_1}(f) = \deg_{x_1}(F)$, $\deg_{x_2}(f) = \deg_{x_2}(F)$, furthermore $\mathrm{lc}_{x_1}(F_i)$

Output: F_i such that $F = \prod_i^r F_i$ or "failure"

Instructions:

1. **for** $i = 1, \ldots, r$ $h_i = 0$
2. **for** $i = 1, \ldots, r$
   (a) **for** each monomial $x_1^{\nu_i} x_2^{\eta_i}$ occuring in f_i do
      i. $h_i = h_i + x_1^{\nu_i} x_2^{\eta_i} \Lambda_{\nu_i,\eta_i}^{(i)}$
3. $H = \prod_{i=1}^{r} h_i$
4. **if** there are monomials $x_1^{\alpha} x_2^{\beta}$ not occuring in both F and H **return** "failure"
5. **for** each monomial $x_1^{\alpha} x_2^{\beta}$ occuring in F and H build an equation $A^{(\alpha,\beta)} = F^{(\alpha,\beta)} - H^{(\alpha,\beta)}$
6. **for** each $\Lambda_{d_i,\eta_i}^{(i)}$ with $d_i = \deg_{x_1}(f_i)$ plug in the coefficient of $x_2^{\eta_i}$ occurring in $\mathrm{lc}_{x_1}(F_i)$
7. **if** there are equations not yet solved **then**:
   (a) **if** there are no linear equations **then return** "failure"
   (b) **else** solve all linear equations
   (c) plug the new solutions in the remaining equations
   (d) **if** any of the two preceding steps reveals inconsistent solutions **return** "failure"
   (e) **go back** to step 7
8. **else**
   (a) from the solutions for $\Lambda_{\nu_i,\eta_i}^{(i)}$ construct F_i
   (b) **return** $F_1, \ldots, F_r$

Before we prove correctness of the algorithm, we need to introduce the notion of a skeleton [Zip93]: Let $f = \sum_{i=1}^{T} f_i \mathbf{x}^{\alpha_i} \in R[x_1, \ldots, x_n]$ be a polynomial and let T be the number of non-zero terms of f. Then the skeleton $\mathrm{skel}(f) = \{\alpha_1, \ldots, \alpha_T\}$ and let $\mathrm{skel}_k(f)$ be the canonical projection of $\mathrm{skel}(f)$ on the first k components.

Proof: Since it is checked whether there are monomials in x_1 and x_2 which occur in F but not in H and vice versa, it is ensured that $\mathrm{skel}_2(F) = \mathrm{skel}_2(H)$. Now there may be terms in F_i that cancel after multiplying and hence may lead to inconsistent solutions. These are

detected by checking the consistency of the solutions.

Example: Consider $F = F_1F_2F_3 = 57(xyz + 84xy + 28x + 92)(76x^2y + 27x^2 + z + 78)(64y^3 + xz + 71x + 51) \in \mathbb{F}_{101}[x, y, z]$. If we choose z as main variable and y as second variable, we obtain:

$$F(26, y, z) = 80f_1f_2f_3 = 80((z + 84)y + 16)(y + 52z + 23)(y^3 + 73z + 47).$$

The leading coefficients of F_1, F_2,F_3 are precomputed as $35x$, 52, $30x$. Then:

$$\left(\begin{array}{c}
\Lambda^{(1)}_{0,0}\Lambda^{(2)}_{0,0}\Lambda^{(3)}_{0,0} + 27x^4 + 76x^3 + 96x^2 + 40x + 52 \\
(\Lambda^{(1)}_{0,0}\Lambda^{(2)}_{0,1} + \Lambda^{(1)}_{0,1}\Lambda^{(2)}_{0,0})\Lambda^{(3)}_{0,0} + 56x^4 + 16x^3 + 49x^2 + 87x \\
\Lambda^{(1)}_{0,1}\Lambda^{(2)}_{0,1}\Lambda^{(3)}_{0,0} + 26x^4 + 77x^3 \\
\Lambda^{(1)}_{0,0}\Lambda^{(2)}_{0,0}\Lambda^{(3)}_{0,3} + 3x^3 + 82x^2 + 76x + 91 \\
(\Lambda^{(1)}_{0,0}\Lambda^{(2)}_{0,1} + \Lambda^{(1)}_{0,1}\Lambda^{(2)}_{0,0})\Lambda^{(3)}_{0,3} + 96x^3 + 55x^2 + 26x \\
\Lambda^{(1)}_{0,1}\Lambda^{(2)}_{0,1}\Lambda^{(3)}_{0,3} + 59x^3 \\
\Lambda^{(1)}_{0,0}\Lambda^{(2)}_{0,0}\Lambda^{(3)}_{1,0} + \Lambda^{(1)}_{0,0}\Lambda^{(2)}_{1,0}\Lambda^{(3)}_{0,0} + 90x^4 + 36x^3 + 59x^2 + 32x + 68 \\
(\Lambda^{(1)}_{0,0}\Lambda^{(2)}_{0,1} + \Lambda^{(1)}_{0,1}\Lambda^{(2)}_{0,0})\Lambda^{(3)}_{1,0} + (\Lambda^{(1)}_{0,1}\Lambda^{(2)}_{1,0} + \Lambda^{(1)}_{1,1}\Lambda^{(2)}_{0,0})\Lambda^{(3)}_{0,0} + 71x^4 + 32x^3 + 86x^2 + 89x \\
\Lambda^{(1)}_{0,1}\Lambda^{(2)}_{0,1}\Lambda^{(3)}_{1,0} + \Lambda^{(1)}_{1,1}\Lambda^{(2)}_{0,1}\Lambda^{(3)}_{0,0} + 99x^4 + 43x^3 \\
\Lambda^{(1)}_{0,0}\Lambda^{(2)}_{1,0}\Lambda^{(3)}_{0,3} + 45x + 18 \\
(\Lambda^{(1)}_{0,1}\Lambda^{(2)}_{1,0} + \Lambda^{(1)}_{1,1}\Lambda^{(2)}_{0,0})\Lambda^{(3)}_{0,3} + 47x^3 + 80x \\
\Lambda^{(1)}_{1,1}\Lambda^{(2)}_{0,1}\Lambda^{(3)}_{0,3} + 50x^3 \\
\Lambda^{(1)}_{0,0}\Lambda^{(2)}_{1,0}\Lambda^{(3)}_{1,0} + 37x^2 + 35x \\
(\Lambda^{(1)}_{0,1}\Lambda^{(2)}_{1,0} + \Lambda^{(1)}_{1,1}\Lambda^{(2)}_{0,0})\Lambda^{(3)}_{1,0} + \Lambda^{(1)}_{1,1})\Lambda^{(2)}_{1,0}\Lambda^{(3)}_{0,0} + 97x^4 + 59x^2 + 71x \\
\Lambda^{(1)}_{1,1}\Lambda^{(2)}_{0,1}\Lambda^{(3)}_{1,0} + 86x^4 \\
\Lambda^{(1)}_{1,1}\Lambda^{(2)}_{1,0}\Lambda^{(3)}_{0,3} + 99x \\
\Lambda^{(1)}_{1,1}\Lambda^{(2)}_{1,0}\Lambda^{(3)}_{1,0} + 41x^2
\end{array}\right)$$

After plugging in the precomputed values for the leading coefficients the above system looks like follows:

$$\left(\begin{array}{c}
\Lambda^{(1)}_{0,0}\Lambda^{(2)}_{0,0}\Lambda^{(3)}_{0,0} + 27x^4 + 76x^3 + 96x^2 + 40x + 52 \\
(\Lambda^{(1)}_{0,0}\Lambda^{(2)}_{0,1} + \Lambda^{(1)}_{0,1}\Lambda^{(2)}_{0,0})\Lambda^{(3)}_{0,0} + 56x^4 + 16x^3 + 49x^2 + 87x \\
\Lambda^{(1)}_{0,1}\Lambda^{(2)}_{0,1}\Lambda^{(3)}_{0,0} + 26x^4 + 77x^3 \\
\Lambda^{(1)}_{0,0}\Lambda^{(2)}_{0,0}\Lambda^{(3)}_{0,3} + 3x^3 + 82x^2 + 76x + 91 \\
(\Lambda^{(1)}_{0,0}\Lambda^{(2)}_{0,1} + \Lambda^{(1)}_{0,1}\Lambda^{(2)}_{0,0})\Lambda^{(3)}_{0,3} + 96x^3 + 55x^2 + 26x \\
\Lambda^{(1)}_{0,1}\Lambda^{(2)}_{0,1}\Lambda^{(3)}_{0,3} + 59x^3 \\
52\Lambda^{(1)}_{0,0}\Lambda^{(2)}_{0,0} + 30x\Lambda^{(1)}_{0,0}\Lambda^{(2)}_{0,0} + 90x^4 + 36x^3 + 59x^2 + 32x + 68 \\
(35x\Lambda^{(2)}_{0,0} + 52\Lambda^{(1)}_{0,1})\Lambda^{(2)}_{0,0} + 30x\Lambda^{(1)}_{0,0}\Lambda^{(2)}_{0,1} + 30x\Lambda^{(1)}_{0,1}\Lambda^{(2)}_{0,0} + 71x^4 + 32x^3 + 86x^2 + 89x \\
35x\Lambda^{(2)}_{0,1}\Lambda^{(2)}_{0,0} + 30x\Lambda^{(1)}_{0,1}\Lambda^{(2)}_{0,1} + 99x^4 + 43x^3 \\
52\Lambda^{(1)}_{0,0}\Lambda^{(3)}_{0,3} + 45x + 18 \\
(35x\Lambda^{(2)}_{0,0} + 52\Lambda^{(1)}_{0,1})\Lambda^{(3)}_{0,3} + 47x^3 + 80x \\
35x\Lambda^{(2)}_{0,1}\Lambda^{(3)}_{0,3} + 50x^3 \\
45x\Lambda^{(1)}_{0,0} + 37x^2 + 35x \\
2x\Lambda^{(2)}_{0,0} + 40x^2\Lambda^{(2)}_{0,0} + 45x\Lambda^{(1)}_{0,1} + 97x^4 + 59x^2 + 71x \\
40x^2\Lambda^{(2)}_{0,1} + 86x^4 \\
2x\Lambda^{(3)}_{0,3} + 99x \\
0
\end{array}\right)$$

Now one can read off the solutions for $\Lambda^{(1)}_{0,0} = 71x + 89$, $\Lambda^{(2)}_{0,1} = 13x^2$, and $\Lambda^{(3)}_{0,3} = 1$. Substituting these solutions back, one eventually arrives at $\Lambda^{(1)}_{0,1} = 11x$, $\Lambda^{(2)}_{0,0} = 91x^2 + 16$, and $\Lambda^{(3)}_{0,0} = 9x + 15$. The final solution then reads: $F_1 = 35xyz + 11xy + 71x + 89$, $F_2 = 52z + 13x^2y + 91x^2 + 16$, and $F_3 = 30xz + y^3 + 9x + 15$.

Obviously, this method fails if F has terms that vanish after substituting $b_3, \ldots, b_n$ for $x_3, \ldots, x_n$. By [Zip93, Proposition 116] this has probability $\frac{(n-2)(n-3)DT}{B}$ if one chooses a point $b_3, \ldots, b_n$ in S^{n-2}, where S is a subset of R of cardinality B, $\deg_{x_i} < D$ for $i \geq 3$, and T denotes the number of terms. On the other hand, it fails if there is term-cancellation when multiplying the factors.

We now present a genuine idea to lift very sparse polynomials:
Assume a polynomial $F \in R[x_1, x_2, \ldots, x_n]$, an evaluation point

$a = (a_2, \ldots, a_n)$, and factorizations

$$f^{(j)} = F(x_1, a_2, \ldots, a_{j-1}, x_j, a_{j+1}, \ldots, a_n) = \prod_{i=1}^{r_j} f_i^{(j)}$$

for all $2 \leq j \leq n$ such that $\deg_{x_1}\left(f^{(j)}\right) = \deg_{x_1}(F)$ and $r_k = r_l$ for all $k \neq l$ are given. Furthermore, let us assume that no extraneous factors occur in the bivariate factorizations, and that the number of terms $t_i^{(j)}$ in each factor $f_i^{(j)}$ coincides for all $2 \leq j \leq n$ and equals the number of terms t_i in the factors F_i of F. Note that $f^{(k)}(a_k) = f^{(l)}(a_l)$ for all $k \neq l$. If one now splits each $f_i^{(j)}$ into terms and evaluates each of these terms separately at a_j, one obtains lists $l_i^{(j)}$ of polynomials in $R[x_1]$. Given the entries in $l_i^{(j)}$ are all distinct for all i and j, one can reconstruct the terms of F_i by comparing the entries of $l_i^{(j)}$ in the following way: For fixed i, one initializes h_0 to an entry of $l_i^{(2)}$, which has not been considered yet, and h_1 to the corresponding monomial in $f_i^{(2)}$. Now h_0 occurs among the entries of $l_i^{(j)}$ because $t_i^{(j)} = t_i$ for all j. For $3 \leq j \leq n$, one computes h_{j-1} as the least common multiple of h_{j-2} and the to h_0 corresponding monomial of $f_i^{(j)}$. Eventually, h_{n-1} equals a monomial which occurs in F_i. The coefficient of the corresponding term can be determined as $h_0/h_{n-1}(a)$. One can finally determine all terms of all F_i by repeating the above procedure for all i and all entries in $l_i^{(2)}$.

Example: Let

$$F = F_1 F_2 = (-xy + 3xz + 3)(xyz - 3x + y + 3z) \in \mathbb{Q}[x, y, z].$$

Choosing z as main variable, one obtains

$$F(x, -5, z) = f_1^{(x)} f_2^{(x)} = (-3xz - 5x - 3)(5xz + 3x - 3z + 5)$$

and

$$F(-22, y, z) = f_1^{(y)} f_2^{(y)} = (-22y + 66z - 3)(22yz - y - 3z - 66).$$

Then

$$\begin{aligned} l_1^{(x)} &= \{66z, 110, -3\}, \\ l_1^{(y)} &= \{110, 66z, -3\}, \\ l_2^{(x)} &= \{-110z, -66, -3z, 5\}, \\ l_2^{(y)} &= \{-110z, 5, -3z, -66\}. \end{aligned}$$

Now one sets $h_0 = 66z$ and $h_1 = xz$. Then $h_2 = \mathrm{lcm}(xz, z) = xz$. The proper coefficient is computed as $h_0/h_2(-22, -5) = 66/-22 = -3$. Repeating this procedure yields $F_1 = -3xz + xy - 3$ and $F_2 = -xyz + 3x - 3z - y$.

Requiring the number of terms $t_i^{(j)}$ of the bivariate factors to coincide mutually and to coincide with the number of terms t_i of the multivariate factors is a very strong restriction. Even though, the condition is not met, one may still deduce a factor of F by examining its bivariate factorizations as the next example shows:

Example: Let
$F = F_1F_2 = (5xy^5 + 11s^5 + y^2z + 11yz + 8s + 12)(16xz^3s^2 + 11y^4z + 9xz^2s + 7xys + 13xz + 12y + 6) \in \mathbb{F}_{17}[x, y, z, s]$.
Choosing y as main variable one obtains

$$\begin{aligned} F(x, y, 14, 10) =& (5xy^5 + 14y^2 + y + 3)(y^4 + 2xy + 3x + 12y + 6), \\ F(2, y, z, 10) =& (10y^5 + y^2z + 11yz + 3)(11y^4z + 4z^3 + 10z^2 + 16y + 9z + 6), \\ F(2, y, 14, s) =& (10y^5 + 11s^5 + 14y^2 + y + 8s + 10)(y^4 + 14ys + 3s^2 + 12y + 9s + 13). \end{aligned}$$

Then

$$\begin{aligned} l_2^{(x)} =& \{y^4, 4y, 6, 12y, 6\}, \\ l_2^{(z)} =& \{y^4, 11, 5, 16y, 7, 6\}, \\ l_2^{(s)} =& \{y^4, 4y, 11, 12y, 5, 13\}. \end{aligned}$$

As before, we find y^4 in each $l_2^{(j)}$ and conclude that $11y^4z$ occurs as term in F_2. Now $4y$ and $12y$ can only be found in $l_2^{(s)}$ and $l_2^{(s)}$, but since $f_2^{(k)}(a_k) = f_2^{(l)}(a_l)$, we conclude that $7xys + 12y$ occurs in F_2. To deduce the remaining terms of F_2, note that F contains a constant term and hence F_1 and F_2 do, too. Therefore, we conclude that the last entry 6 of $l_2^{(x)}$ corresponds to this constant term as the first 6 corresponds to the monomial x. Since 11 and 5 occur in both $l_2^{(z)}$ and $l_2^{(s)}$, we assume that F_2 contains a part that looks like $az^3s^2 + bz^2s$ for some $a, b \in \mathbb{F}_{17}[x]$. Examining $l_2^{(z)}$, we find that $11 + 5 + 7 = 6$ and hence conclude that F_2 contains a part that looks like $cxz^3 + dxz^2 + exz$ for some $c, d, e \in \mathbb{F}_{17}[s]$. Comparing these two equations, we deduce the following equation: $fxz^3s^2 + gxz^2s + hxz$ for some $f, g, h \in \mathbb{F}_{17}$. Now f, g, h can be determined as $f = 11/(2 \cdot 14^3 \cdot 10^2) = 16$, $g = 5/(2 \cdot 14^2 \cdot 10) = 9$, $h = 7/(2 \cdot 14) = 13$. In all, this yields a tentative factor which looks like $16xz^3s^2 + 11y^4z + 9xz^2s + 7xys + 13xz + 12y + 6$ and, in fact, it divides F.

6.4 Leading coefficient precomputation

Precomputed leading coefficients are necessary in the bivariate to multivariate lifting since if requiring the factors to be monic in x_1, the Hensel lifting would compute $\mathrm{lc}_{x_1}(F_i)^{-1}F_i \bmod \left(x_2^{k_2}, \ldots, x_n^{k_n}\right)$ which leads to tremendous coefficient swell. Another solution is to impose $\mathrm{lc}_{x_1}(F)$ as leading coefficient of each factor and multiply F by $\mathrm{lc}_{x_1}(F)^{r-1}$. This is disadvantageous as sparseness of F may be destroyed, too. Hence, it is desirable to apply a multiplier as small as possible on F.

There are two methods to precompute leading coefficients, one is due to Wang [Wan78], another is due to Kaltofen [Kal85c]. Wang's method can be used over $\mathbb{Z}$ and $\mathbb{Q}(\alpha)$, and also in the case where one lifts from univariate to bivariate polynomials, whereas Kaltofen's method works over any UFD but requires bivariate factors. Both require a one-to-one correspondence of univariate to bivariate factors and bivariate to multivariate factors respectively. As we have seen, one-to-one correspondence of bivariate to multivariate factors is very likely, but a one-to-one correspondence of univariate to bivariate factors is only likely in case $R = \mathbb{Z}$ or $\mathbb{Q}(\alpha)$ or more general R is Hilbertian. Even though, Wang's method eventually produces the correct leading coefficients, and Kaltofen's method probably only a partial solution, Wang's method requires an irreducible factorization of the leading coefficient and may need "large" evaluation points. Computing the irreducible factorization of the leading coefficient is quite expensive, and using "large" evaluation points makes subsequent steps more expensive. Therefore, we did not use Wang's method.

Example: Consider $f = (x^2y^7 + 1)(64x^3y^3 + 7)$. Then $\mathrm{lc}_x(f) = 64y^{10}$. The smallest evaluation point that is feasible for Wang's method is 4. Hence, one needs to factorize $67108864x^5 + 4096x^3 + 114688x^2 + 7$. In contrast, if one does not use Wang's method, 1 is a feasible evaluation point and one needs to factorize $64x^5 + 64x^3 + 7x^2 + 7$.

In Kaltofen's method [Kal85c], the precomputation of leading coefficients is achieved as follows:

Let $F \in R[x_1, \ldots, x_n]$, $a = (a_3, \ldots, a_n)$ in R^{n-2} be such that:

$$\begin{aligned} F &= F_1 \cdots F_r, \\ F(x_1, x_2, a_3, \ldots, a_n) &= f = f_1 \cdots f_r, \\ F_i(a) &= f_i, \\ \mathrm{lc}_{x_1}(F)(a) &\neq 0, \\ \deg_{x_i}(F) &= \deg_{x_i}(F(x_1, x_2, \ldots, x_i, a_{i+1}, \ldots, a_n)). \end{aligned} \tag{6.4.1}$$

Furthermore, let l denote $\mathrm{lc}_{x_1}(F), l_i = \mathrm{lc}_{x_1}(F_i)$, $l' = \mathrm{lc}_{x_1}(f)$, and $l'_i = \mathrm{lc}_{x_1}(f_i)$. Now let

$$\mathrm{sqrf}(l) = \prod_{(s_i, m_i) \in \mathrm{Sqr}(l)} s_i$$

be the square-free part of l such that

$$(\mathrm{sqrf}(l))(a) = \mathrm{sqrf}(l(a))$$

The aim is to use Hensel lifting to lift the square-free factors of each l'_i, but for that one needs coprimality. Therefore, a so-called *GCD-free basis* is introduced:

Definition 6.1. Let $A = \{a_1, \ldots, a_r\} \subset R \backslash \{0\}$, R a ring. A set $B = \{b_1, \ldots, b_s\}$ is called a GCD-free basis of A if:

1. $\gcd(b_i, b_j) = 1$ for all $i = 1, \ldots, s$
2. for all $i = 1, \ldots, r$ and $j = 1, \ldots, s$ there exists $e_{ij} \in \mathbb{N}$ such that

$$a_i = u \prod_{j=1}^{s} b_j^{e_{ij}}$$

   for some unit $u \in R$

To achieve uniqueness up to multiplication with units, one needs the notion of a *standard* GCD-free basis:

Definition 6.2. A GCD-free basis B of $A \subset R\{0\}$ is called *standard* if:

1. $s = |B| = \min\{|C| \mid \text{C is a GCD-free basis of A}\}$
2. for all $j = 1, \ldots, s$ and $m > 1$: $\{b_1, \ldots, b_{j-1}, b_j^m, b_{j+1}, \ldots, b_s\}$ is not a GCD-free basis of A

Now for each l'_i its square-free decomposition is computed. Afterwards a standard GCD-free basis B of all square-free decompositions is computed. Since all elements of B are pairwise co-prime and $\mathrm{sqrf}(l)(a) = \prod_{b\in B} b$, one can apply Hensel lifting to it.

The result is

$$\mathrm{sqrf}(l) = \prod_{b'|b'(a)\in B} b'$$

Now if $b'(a)$ occurs in $\mathrm{Sqr}(l'_i)$ with exponent m_i, (b', m_i) is part of $\mathrm{Sqr}(l_i)$.

Algorithm 6.11 (Leading coefficient precomputation)**.**
Input: $F, f_1, \ldots, f_r$, and a satisfying the conditions in 6.4.1

Output: a list $m, p_1, \ldots, p_r$ of polynomials such that $m \prod_{i=1}^{r} p_i = l$ and $p_i \mid l_i$

Instructions:

1. compute the square-free part $\mathrm{sqrf}(l)$ of $l = \mathrm{lc}_{x_1}(F)$
2. **if** $\mathrm{sqrf}(l(a)) \neq \mathrm{sqrf}(l)(a)$ **then**
   (a) $m = l$
   (b) $p_i = 1$
   (c) **return** m and $p_1, \ldots, p_r$
3. **for** $i = 1, \ldots, r$ compute the square-free decompositions $\mathrm{Sqr}(l'_i)$ of $l'_i = \mathrm{lc}_{x_1}(f_i)$
4. **for** $i = 1, \ldots, r$ form the set S consisting of all s_{ij} such that $(s_{ij}, m_j) \in \mathrm{Sqr}(l'_i)$ for some m_j
5. compute a minimal GCD-free basis B of S
6. use Hensel lifting to lift the elements b in B from $R[x_2]$ to $R[x_2, \ldots, x_n]$, yielding b' such that $b'(a) = b$
7. **for** $i = 1, \ldots, r$ $p_i = 1$
8. **for** all b' and $i = 1, \ldots, r$
   (a) **if** $(b'(a), m_j) \in \mathrm{Sqr}(l'_i)$ for some m_j **then**

i. $p_i = p_i b^{m_j}$

9. $m = l / \prod_{i=1}^{r} p_i$
10. **return** m and $p_1, \ldots, p_r$

Proof: To prove correctness, it suffices to show that $\prod_{b \in B} b = \mathrm{sqrf}(l(a))$. A square-free factor s of l' is of the form

$$\prod_{i=1}^{r} \prod_{(s_{ij}, m_j) \in \mathrm{Sqr}(l'_i)} s_{ij} / \prod_{k \neq m} \gcd(s_{kv}, s_{mu})$$

Since B is a minimal GCD-free basis of all $\mathrm{Sqr}(l'_i)$, the claim follows.

The choice of x_1 and x_2 in the above description is just for convenience. If possible, they should be chosen such that $\deg_{x_2}(l) > 0$, $\mathrm{cont}_{x_2}(l) = 1$, and $\mathrm{sqrf}(l)(a) = \mathrm{sqrf}(l')$. In the factorization algorithm a main variable x_1 is fixed. Hence, one can only alter the choice of x_2.

Kaltofen argues that using the square-free part may result in an exponential growth of terms and therefore does not use it [Kal85c, Remark 1]. However, in our experiments we never experienced this kind of bad behavior.

Kaltofen's original method uses a recursion in algorithm 6.11 to also distribute m. However, in same cases this recursion may be abridged, or may fail as the last condition of 6.4.1 may not be met or $(\mathrm{sqrf}(l))(a) \neq \mathrm{sqrf}(l(a))$. Therefore, we propose the following heuristics to make m smaller:

Heuristic 1: If $\mathrm{cont}_{x_2}(l) \neq 1$, the above algorithm is not able to distribute $\mathrm{cont}_{x_2}(l)$ on the f_i's. Let us consider $\mathrm{cont}_{x_2}(l)$ as element of $R[x_j][x_3, \ldots, x_{j-1}, x_{j+1}, \ldots, x_n]$. If $\mathrm{cont}(\mathrm{cont}_{x_2}(l)) \neq 1$ and given a bivariate irreducible decomposition of

$$F(x_1, a_2, \ldots, a_{j-1}, x_j, a_{j+1}, \ldots, a_n) = f_1^{(j)} \cdot \ldots \cdot f_r^{(j)}$$

it is possible to distribute a divisor of $\mathrm{cont}(\mathrm{cont}_{x_2}(l))$ on the $f_i^{(j)}$'s (and hence on the f_i's) by considering $\gcd(\mathrm{lc}_{x_1}(f_i^{(j)}), \mathrm{cont}(\mathrm{cont}_{x_2}(l)))$. This heuristic may be used as part of algorithm 6.11.

Heuristic 2: Assume algorithm 6.11 returned an $m \neq 1$ and we have run the heuristic sparse Hensel lifting by Lucks with the output of algorithm

6.11, but it was unable to find more linear equations. It therefore failed, but recovered at least one coefficient a besides the leading coefficient. And let $h_1, \ldots, h_r$ be the output of the sparse Hensel lifting. Then there is at least one h_i which looks like follows:

$$x_1^{d_1} p_i m + \ldots + x_1^{\eta} a + \ldots \tag{6.4.2}$$

with non-zero a.
Now if $\mathrm{cont}_{x_1}(h_i) = 1$, m does not occur in the leading coefficient of F_i.

Heuristic 3: Under the assumptions of heuristic 2: If $\mathrm{cont}_{x_1}(h_i) \neq 1$ for all i, but at least one coefficient besides the leading coefficient has been reconstructed in each factor, and $\prod_i^r \mathrm{lc}_{x_1}(\mathrm{pp}_{x_1}(h_i)) = \mathrm{lc}_{x_1}(F)$. Then one can try to lift the f_i with the obtained leading coefficients.
If there is a factor in which only the leading coefficient has been reconstructed, one can still try to use $\mathrm{lc}_{x_1}(\mathrm{pp}_{x_1}(h_i))$ of those factors of the form 6.4.2 and combine the current heuristic with the following heuristic:

Heuristic 4: Assume $m \neq 1$ and bivariate irreducible decompositions of

$$F(x_1, a_2, \ldots, a_{j-1}, x_j, a_{j+1}, \ldots, a_n) = f_1^{(j)} \cdot \ldots \cdot f_r^{(j)}$$

such that $\deg_{x_1}(F) = \deg_{x_1}(F(x_1, a_2, \ldots, a_{j-1}, x_j, a_{j+1}, \ldots, a_n))$ for all j with $\deg_{x_j}(m) > 0$ are given. Then one can try to determine the variables which occur in $\mathrm{lc}_{x_1}(F_i)$ by examining the given bivariate irreducible decompositions. If there is a square-free factor of m whose variables do not occur among the variables of $\mathrm{lc}_{x_1}(F_i)$, then it does not contribute to $\mathrm{lc}_{x_1}(F_i)$. Note that instead of the square-free decomposition of m one could also consider the irreducible decomposition of m. This heuristic is independent of the result of algorithm 6.10, but we found it to be more error-prone than heuristic 2 and 3 since it requires
$\deg_{x_j}(\mathrm{lc}_{x_1}(F(x_1, a_2, \ldots, a_{j-1}, x_j, a_{j+1}, \ldots, a_n))) > 0$, whenever
$\deg_{x_j}(\mathrm{lc}_{x_1}(F)) > 0$. To discard wrong results, one can check if the product of the obtained leading coefficients is a multiple of the leading coefficient of F.

Chapter 7

Recombination

Since modular and bivariate factors do not need to be in one-to-one correspondence, the lifted modular factors need to be recombined to yield true bivariate factors.

7.1 Naive recombination

A naive recombination algorithm tests each combination of factors by brute force. Nevertheless, an easy way to reduce the number of combinations to check is given in [Ber99]: From the univariate factorizations one can deduce a set of possible degrees in x of the true factors, called possible *degree pattern*. For example, consider a polynomial of degree 8 in x whose modular factors have the following degrees in x: $(2,3,3)$,$(1,2,5)$, and $(4,4)$. From the first set, we conclude that a true factor can have the following degrees in x: $\{2,3,5,6,8\}$, the second gives $\{1,2,3,5,6,7,8\}$, and the third gives $\{4,8\}$. Now as the degree in x of any true factors has to occur in each set, we can intersect them, which gives $\{8\}$, and conclude that the polynomial in question is irreducible. One can also consider the sets $\{(2,3,3),(2,6),(3,5),8\}$, $\{(1,2,5),(1,7),(2,6),(3,5),8\}$, and $\{(4,4),8\}$ of all possible combinations of degrees. In the worst-case these contain 2^d elements, where d is the degree of f in x, and do not help to exclude any combinations. Hence, we do not use them.

Another way to exclude combinations is to test divisibility at certain points. We propose 0 and 1 and call these tests $0-1$ tests: For this, plug in 0 and/or 1 for x in the candidate factor and test for divisibility. This way, one just has to check divisibility of a univariate polynomial to exclude a bad combination, which is much cheaper than a multivariate divisibility

test. From our experience the $0-1$ tests almost always detect a bad combination.

One can also adapt the $d-1$ and $d-2$ tests, proposed in [ASZ00], to the bivariate case by using Newton polygons. However, due to lack of time and the use of a polynomial time recombination algorithm which can be used if the combinations of a few factors did not yield a complete factorization, we refrained from implementing these tests.

Since the result of the bivariate Hensel lifting 6.1 is polynomials in $R[x][y]/y^n$ that are monic with respect to x, one needs to reconstruct leading coefficients of the factors. For this, one needs to note that a factor g of f has a leading coefficient that divides the leading coefficient of f. After the Hensel lifting one has reconstructed $\mathrm{lc}_x(g)^{-1}g \bmod y^n$. Now multiplying by $\mathrm{lc}_x(f)$ in $R[x][y]/y^n$ and taking the primitive part with respect to x in $R[x][y]$ yields g if n was big enough. Since neither $\deg_y(g)$ nor $\deg(\mathrm{lc}_x(f))$ exceed $\deg_y(f)$, choosing n as $\deg_y(f)+1$ suffices to reconstruct any factor of f.

Algorithm 7.1 (Naive factor recombination)**.**
Input: $f \in R[x][y]$ square-free such that $\mathrm{lc}_x(f)(0) \neq 0$ and $f(x,0)$ is square-free, $f_i^{(n)} \in \mathrm{Quot}(R)[x][y]$ such that $f \equiv \prod_{i=1}^{r} f_i^{(n)} \bmod y^{n+1}$, $n = \deg_y(f)$ and $f_i^{(n)} \equiv f_i \bmod y$ for all $i = 2, \ldots, r$, where f_i are the monic irreducible factors of $f(x,0)$ in $\mathrm{Quot}(R)[x]$, and $f_1 = \mathrm{lc}_x(f)$

Output: all irreducible factors $g_1, \ldots, g_s \in R[x][y]$ of f

Instructions:

1. $T = \{f_2^{(n)}, \ldots, f_r^{(n)}\}$, $s = 1$, $G = \emptyset$, $h = f$
2. **while** $T \neq \emptyset$
   (a) **for** all subsets S of T of size s
      i. $g = \mathrm{lc}_x(h) \prod_{f_i^{(n)} \in S} f_i^{(n)} \bmod y^{n+1}$
      ii. $g = \mathrm{pp}_x(g)$
      iii. **if** $g \mid h$
         A. $h = h/g$
         B. $T = T \backslash S$
         C. $G = G \cup \{g\}$

(b) $s = s + 1$

(c) **if** $2s > |T|$

i. $g = \mathrm{lc}_x(h) \prod_{f_i^{(n)} \in T} f_i^{(n)} \bmod y^{n+1}$

ii. $g = \mathrm{pp}_x(g)$

iii. $G = G \cup \{g\}$

iv. **return** G

Proof: Since the lifting precision n suffices to reconstruct any factor of f and any possible combination of lifted factors is checked, the algorithm is correct.

In essence, the naive recombination algorithm 7.1 is an adaption of [vzGG03, Algorithm 15.22], tough we use a lower lifting precision.

For the degree pattern test one passes a set of possible degrees dp as additional input, and checks in step 2a if $\sum_{f_i^{(n)} \in S} \deg_x(f_i^{(n)})$ occurs in dp. Every time a factor g of f is found, one needs to update dp as $\mathrm{dp} = \mathrm{dp} \cap \mathrm{dp}_{T \setminus S}$ before step 2(a)iiiB. If $|\mathrm{dp}| = 1$, h is irreducible.

The 0-1 tests can be applied before step 2(a)i.

If one has passed to a field extension $\mathbb{F}_{q^k}$ of $\mathbb{F}_q$, one first can check if g is an element of $\mathbb{F}_q[x][y]$ before attempting any divisibility testing.

Remark 7.1. If one factorizes a polynomial over $\mathbb{Q}$, the factors $f_i^{(n)}$ passed to algorithm 7.1 by algorithm 8.2 are polynomials in $(\mathbb{Z}/p^k\mathbb{Z})[x][y] \bmod y^{n+1}$ that are monic with respect to x and $p^k > 2B$, where B is a bound on the absolute value of the coefficients of a factor g of f. The input f is in $\mathbb{Z}[x][y]$. In this case, one needs to alter step 2(a)i in algorithm 7.1 to:

$$g = \mathrm{lc}_x(h) \prod_{f_i^{(n)} \in S} f_i^{(n)} \bmod \ < y^{n+1}, p^k >$$

where 'mod p^k' means symmetric remaindering here.

Remark 7.2. If one factorizes a polynomial over $\mathbb{Q}(\alpha)$, the factors $f_i^{(n)}$ passed to algorithm 7.1 by algorithm 8.3 are polynomials in $(\mathbb{Z}/p^k\mathbb{Z})[t]/(\mu_p)[x][y] \bmod y^{n+1}$. The input f is in $\mathbb{Z}[\alpha][x][y]$. In addition to

the above case, also the possible denominators $d_f D$ of a factor are passed. Hence, one needs to alter step 2(a)i to:

$$g = d_f D \cdot \mathrm{lc}_x(h) \prod_{f_i^{(n)} \in S} f_i^{(n)} \bmod \ < y^{n+1}, p^k >$$

Again, 'mod p^k' means symmetric remaindering here. Clearing the denominators is necessary because reduction modulo μ may introduce denominators to $\mathrm{lc}_x(h) \prod_{f_i^{(n)} \in S} f_i^{(n)}$.

If there are irreducible factors g of f whose reduction $g(x, a)$ remains irreducible in $R[x]$, one may be able to reconstruct them before reaching precision $\deg_y(f) + 1$ in the Hensel lifting. We call this *early* factor reconstruction. To find promising lifting precisions for conducting a successful early factor reconstruction, one may factorize the leading coefficient of f and stop the lifting process at every degree of a possible factor of the leading coefficient. In the worst case, this means after *every* Hensel step and furthermore requires a relatively expensive factorization of the leading coefficient. So, instead we propose to take the shape of the Newton polygon of f into account.

Definition 7.1. The Newton polygon $\mathcal{N}(f)$ of a polynomial $f = \sum a_{ij} x^i y^j \in R[x, y]$ is the convex hull of all $(i, j) \in \mathbb{N}^2$ for which $a_{ij} \neq 0$.

A classic theorem by Ostrowski relates the decomposition of a polynomial to the decomposition of its Newton polygon as Minkowski sum [Ost99]:

Theorem 7.1. *Let $f, g, h \in R[x, y]$ such that $f = gh$ then $\mathcal{N}(f) = \mathcal{N}(g) + \mathcal{N}(h)$.*

One can show that each face of $\mathcal{N}(f)$ is a sum of faces of $\mathcal{N}(g)$ and $\mathcal{N}(h)$[Grü67] .

Considering all edges $e = (u, v)$ of $\mathcal{N}(f)$ with positive slope in y-axis direction, that is, $u_2 - v_2 > 0$ (when passing through $\mathcal{N}(f)$ counter-clockwise), we stop the lifting at all combinations of $u_2 - v_2$ and perform an early factor reconstruction. If the number of combinations exceeds a certain threshold or if there is only one edge e with the above property, we stop the lifting at $1/4 \deg_y(f)$, $1/2 \deg_y(f)$, and $3/4 \deg_y(f)$.

7.2 Polynomial time recombination

One of the first polynomial time recombination algorithms over a finite field is due to Lenstra [Len83a]. It reduces a lattice in $\mathbb{F}_q[x]^{m+1}$ for $m = \deg_y(h), \deg_y(h)+1, \ldots, \deg_y(f)-1$, where h is an irreducible factor of f mod y, and needs the modular factors to be lifted up to precision $k = 2\deg_y(f)\deg_x(f)+1$. Since reducing such a large lattice and lifting up to such high precision is very expensive; this method was almost impracticable and only useful in very rare cases.

We follow the ideas of [BvHKS09] and assume that f is square-free. Now each true bivariate factor g_j of f is equal to a product of modular factors f_i lifted to high enough precision. Or more mathematically:

$$g_j = \prod \mathrm{lc}_x(g_j) f_i^{\mu_{ji}} \bmod y^k,$$

where μ_{ji} is an element in $\{0,1\}$. To determine the true factors, one has to determine the μ_{ji}. This is also called knapsack factorization. The $\mu_j = (\mu_{j1}, \ldots, \mu_{jr})$ span a lattice W over $\mathbb{F}_p$, which we call *target lattice*. And any lattice with entries in $\{0,1\}$ and exactly r entries equal to 1 is called *reduced*.

The aim is to compute lattices $L' \subset L = \mathbb{F}_p^r$ that contain W such that eventually $L' = W$ holds.
To do so, linearized logarithmic derivatives are used: Let g be a factor of f, then $\Phi(g)$ of g is defined to be fg'/g. One can show the following:

Lemma 7.1. *Let $B_i = \sup\{j \in \mathbb{N} | (i+1, j) \in \mathcal{N}(f)\}$, where $\mathcal{N}(f)$ is the Newton polygon of f, and let $g \in \mathbb{F}_q[y][x]$ be a polynomial that divides f. Then $\Phi(g) = \sum_{i=0}^{n-1} a_i(y)x^i \in \mathbb{F}_q[y][x]$ with*

$$\deg(a_i) \leq B_i.$$

Proof: see [BvHKS09, Lemma 4.3]

Now let $m_i = B_i + 1$ and

$$\Phi(f_j) = \sum_{i=0}^{n-1} a_{ij} x^i \bmod y^l.$$

Furthermore, let A_i be the matrix whose j-th column consists of the coefficients of $y^{m_i}, \ldots, y^{l-1}$ in a_{ij}. This is a $(l - m_i) \times r$ matrix satisfying

$A_i e = 0$ for all $e \in W$ since $\Phi(fg) = \Phi(f) + \Phi(g)$ and the above lemma. Now intersecting the kernels of all A_i gives a lattice L' which contains W.

It is possible to show that if

$$l > \min((2n-1)\deg_y(f), \mathrm{tdeg}(f)(\mathrm{tdeg}(f)-1))$$

then $L' = W$ [BvHKS09, Theorem 4.4].
This gives rise to the following algorithm:

Algorithm 7.2 (Polynomial time recombination)**.**
Input: $f \in \mathbb{F}_q[x][y]$ square-free of degree n in x, $\mathrm{lc}_x(f)(0) \neq 0$, and $f(x,0)$ square-free, $f = \mathrm{lc}_x(f) f_1 \cdot \ldots \cdot f_r \bmod y$ the irreducible decomposition of $f(x,0)$ in $\mathbb{F}_q[x]$

Output: s vectors $\mu_j \in \{0,1\}^r$ such that $g_j = \mathrm{lc}_x(g_j) \prod f_i^{\mu_{ji}} \bmod y^k$, where $f = g_1 \cdot \ldots \cdot g_s$ is the irreducible decomposition of f in $\mathbb{F}_q[x][y]$

Instructions:

1. compute B_i as in Lemma 7.1
2. $l = 1$, $l' = l$, $N = \mathbb{1}_r \in \mathbb{F}_q^{r \times r}$
3. **while**
   (a) $l = l + l'$
   (b) **call** algorithm 6.1 to compute $f = \mathrm{lc}(f) f_1 \cdot \ldots \cdot f_r \bmod y^l$
   (c) **for** $j = 1, \ldots, r$ compute $\Phi(f_j) = \sum_{i=0}^{n-1} a_{ij} x^i \bmod y^l$
   (d) **for** $i \in \{i' \in \{0, \ldots, n-1\} | l - l' \geq B_{i'} + 1\}$
      i. $k_0 = \min(B_i, l - l')$
      ii. write $a_{ij} = \sum_{k=0}^{l-1} c_{kj} y^k$ for $j = 1, \ldots, r$
      iii. $C_i = \begin{pmatrix} c_{k_0 1} & \ldots & c_{k_0 r} \\ \vdots & \ddots & \vdots \\ c_{(l-1)1} & \ldots & c_{(l-1)r} \end{pmatrix} \in \mathbb{F}_q^{(l-k_0)\times r}$
      iv. compute $N' \in \mathbb{F}_q^{d \times d'}$ of maximum rank such that $(C_i N) N' = 0$
      v. $N = NN' \in \mathbb{F}_q^{r \times d}$
   (e) **if** $d = 1$ return N
   (f) **if** $l > \min((2n-1)\deg_y(f), \mathrm{tdeg}(f)(\mathrm{tdeg}(f)-1))$ **break**

(g) $l' = 2l'$

4. **return** the columns of N

If $q = p^m$, then $\mathbb{F}_q$ can be considered as an m-dimensional vector space over $\mathbb{F}_p$ and each element of $\mathbb{F}_q$ can be written as $\sum_{i=0}^{m-1} c_i \alpha^i$. Now if $p > n$, then rewriting each element of $\mathbb{F}_q$ in this way leads to lower lift bounds [BvHKS09, Theorem 4.11]; namely $l > \min(p, n)(n-1)$ instead of $l > \min(q, n)(n-1)$ if we assume $\mathrm{tdeg}(f) = n$. Furthermore, one can show that if $l > 2(n-1)$, then all 0-1 vectors in L' belong to the target lattice W [BvHKS09, Theorem 4.14].

Remark 7.3. In the above algorithm we assumed 0 to be a good evaluation point. This may not always be possible because $\mathbb{F}_q$ may be too small. In this case one has to switch to an extension $\mathbb{F}_q(\alpha)$. Then one can identify $\mathbb{F}_q(\alpha)[\tilde{y}]/(\tilde{y}^l)$ with $\mathbb{F}_q[y]/(\mu^l)$, where μ is the minimal polynomial of α, by setting $\tilde{y} = y - \alpha$. Note that in [BvHKS09] the algorithm is described "mod μ^l", instead of our simpler description with $\mu = y$, and hence all preceding statements remain valid. If one needs to pass to a field extension, the lifting takes place in $\mathbb{F}_q(\alpha)[\tilde{y}]/(\tilde{y}^l)$, and one can divide all bounds from above by $\deg(\mu)$ due to the above identification.
The isomorphism between $\mathbb{F}_q(\alpha)[\tilde{y}]/(\tilde{y}^l)$ and $\mathbb{F}_q[y]/(\mu^l)$ can be implemented using linear algebra. Then the a_{ij} in step 3.(d)ii. can be considered as polynomials in $\tilde{y}$ and α, and therefore by identifying $\tilde{y}$ with $\tilde{y}^{\deg(\mu)}$ and α with $\tilde{y}$, one can map the a_{ij} by simple matrix-vector products.

Since the lattices L' always contain the target lattice W, one can stop as soon as L' is reduced. That is, step 3.(f) can be changed to step 3.(f'): **if** N is reduced **break**. Then the factors lifted so far can be recombined accordingly, and lifted further until they divide f. In case they do not, one can switch to a naive recombination algorithm if the number of factors has already dropped below a certain threshold; or call the above algorithm again, but now use a higher lifting precision.

Even though, the bound on the precision seems to be high, in practice this bound is almost never reached. For example, to factorize the polynomial S_9' from 11.3.3, the precision needed to arrive at the target lattice is 34.
Since the computation of $\Phi(g)$ can become a bottleneck, it makes sense to introduce a threshold depending on the bounds B_i when to use the above algorithm. If the B_i's exceed this threshold, then one can use a naive recombination algorithm, and if checking subsets of factors of size - for

instance $1, 2$ and 3 - did not yield the complete factorization, switch to the above algorithm.

A similar approach as above is taken in [BLS$^+$04],[Lec06], and [Lec10] also using a knapsack and linearized logarithmic derivatives. The best lifting precision that is needed to solve the recombination problem is $\deg_y(f) + 1$ [Lec10, Proposition 4]. However, the bottleneck in practice happens to be Hensel lifting; and the algorithm in [Lec10] does not admit to (partially) solve the recombination problem at low lifting precisions. Hence, the number of factors does not decrease during lifting, which makes it not competitive with [BvHKS09] in general, even though its worst case complexity is better.

Chapter 8

Bivariate factorization

As mentioned before, bivariate factorization is reduced to univariate factorization by plugging in a suitable point for one of the variables. The resulting univariate polynomial is factorized and then y-adic factors are constructed from the univariate factors by Hensel lifting. Afterwards these factors are recombined to yield the true bivariate factors.
We give three versions to describe the full bivariate factorization algorithm: one for bivariate polynomials over finite fields, one over $\mathbb{Q}$, and one over $\mathbb{Q}(\alpha)$. For convenience, we assume that the input polynomial is square-free and primitive with respect to x and y

Algorithm 8.1 (Bivariate polynomial factorization over a finite field)**.**
Input: a square-free polynomial $f \in \mathbb{F}_q[x, y]$ which is primitive with respect to x and y

Output: the irreducible decomposition $\mathrm{Irr}(f)$

Instructions:

1. **if** $\frac{\partial f}{\partial x} = 0$ swap x and y
2. **if** $\deg(\gcd(\frac{\partial f}{\partial x}, f)) > 0$
   (a) **call** the present algorithm with input $g = \gcd(\frac{\partial f}{\partial x}, f)$ and f/g
   (b) **return** $\mathrm{Irr}(f/g) \cup \mathrm{Irr}(g)$
3. choose $a \in \mathbb{F}_q$ at random such that $f(x, a)$ remains square-free and $\deg(f(x, a)) = \deg_x(f)$
4. **if** there is no such a, pass to a suitable field extension $\mathbb{F}_{q^k}$ and **goto** step 3

5. compute $l = \mathrm{Irr}(f(x,a))$ and remove $f_1 = \mathrm{lc}(f(x,a))$ from it
6. $f(x,y) = f(x,y+a)$
7. **call** algorithm 7.2 with input f and l to compute s vectors in $\{0,1\}^r$ such that $g_j = \mathrm{lc}_x(g_j)\prod f_i^{\mu_{ji}} \bmod y^k$ for $j = 1,\ldots,s$, where g_j is an irreducible factor of f over $\mathbb{F}_q$
8. reconstruct the leading coefficients of g_j **for** $j = 1,\ldots,s$
9. $g_j = g_j(x,y-a)$ **for** $j = 1,\ldots,s$
10. **return** $g_1,\ldots,g_s$

Proof: Correctness follows from [vzGG03, Exercise 15.25] as eventually a large enough field extension contains a feasible evaluation point and the correctness of algorithm 7.2.

In [vzGG03, Exercise 15.25] the field is required to contain at least $4\deg_x(f)\deg_y(f)$ elements to prove that only an expected number of 2 values needs to be tested in step 3. However, this theoretic bound relies on estimating the number of roots of a resultant. These bounds are usually very poor. Hence, one should not pass to such large extensions, but try small ones first. If it is necessary to pass to a field extension, one needs to use the modifications of algorithm 7.2 as discussed in remark 7.3

Remark 8.1. When replacing the call to algorithm 7.2 by a call to the naive recombination algorithm 7.1, the above algorithm can be used over any UFD with effective univariate polynomial factorization that contains a feasible evaluation point (see [vzGG03, Algorithm 15.22.]).

In an implementation one should try several different values for a and also try to factorize $f(b,y)$ for random b if possible. This way, one can choose a univariate factorization with few factors and can detect irreducible input before any lifting by computing possible degree patterns. Also note that one should not use the polynomial recombination algorithm if the necessary bounds B_i are large, as otherwise computing the linearized logarithmic derivatives becomes a bottleneck. In this case, one should first use the naive algorithm on subsets of small size. If this does not yield the full factorization, then switching to algorithm 7.2 is usually more favorable.

As mentioned above, since each lattice contains the target lattice W after step 3.(e) of algorithm 7.2, one may leave algorithm 7.2 as soon as N is

reduced. At this point, the lifting precision may not be sufficient to reconstruct the irreducible factors. Hence, one can first recombine the $f_i \in \mathrm{Irr}(f(x,a))$ according to N, and lift the resulting factors to a precision high enough to reconstruct the leading coefficient. If some of the factors now turn out to be no divisors of f in $\mathbb{F}_q[x,y]$, N does not coincide with W. In this case, one can enter algorithm 7.2 again with the remaining factors or try to use the naive algorithm 7.1.

To complete the description, we state the variants over $\mathbb{Q}$ and $\mathbb{Q}(\alpha)$:

Algorithm 8.2 (Bivariate polynomial factorization over $\mathbb{Q}$)**.**
Input: a square-free polynomial $f \in \mathbb{Q}[x,y]$ which is primitive with respect to x and y

Output: the irreducible decomposition $\mathrm{Irr}(f)$

Instructions:

1. compute the lowest common denominator g of the coefficients of f and set $f = gf$
2. choose $a \in \mathbb{Z}$ at random such that $f(x,a)$ remains square-free and $\deg(f(x,a)) = \deg_x(f)$
3. compute $l = \mathrm{Irr}(f(x,a))$, where the irreducible decomposition is computed over $\mathbb{Z}$, and remove $f_1 = \mathrm{cont}(f(x,a))$ from it
4. $f(x,y) = f(x,y+a)$
5. compute a bound B on the coefficients of a factor of f as described in 6.1.3
6. choose $k > 0$ and a prime p such that $p \nmid \mathrm{lc}_x(f)(0)$, $f_i \in l$ are pairwise co-prime in $\mathbb{F}_p$, and $p^k > 2B$
7. **for** $f_i \in l$ set $f_i = \mathrm{lc}(f_i)^{-1} f_i \bmod p^k$
8. $\tilde{f} = (\mathrm{lc}_x(f)(0))^{-1} f \bmod p^k$
9. **call** algorithm 6.4 with input $\tilde{f}$, l, p, k, and $n = \deg_y(f) + 1$ to compute $f_i^{(n)}$ such that $\tilde{f} = \mathrm{lc}_x(\tilde{f}) \prod_i f_i^{(n)} \bmod < y^{n+1}, p^k >$ and $f_i^{(n)}(x,0) \in l$
10. **call** algorithm 7.1 with input f, $f_i^{(n)}$, p, k to compute the irreducible factors $g_1, \ldots, g_s$ of f

11. $g_j = g_j(x, y - a)$ **for** $j = 1, \ldots, s$
12. **return** $g_1, \ldots, g_s$

As $\mathrm{lc}_x(f)(0) = \prod \mathrm{lc}(f_i)$ before step 7 and $p \nmid \mathrm{lc}_x(f)(0)$, step 7 and step 8 cannot fail. Furthermore, note that before step 9 the following holds: $\tilde{f} = \mathrm{lc}_x(\tilde{f}) \prod f_i \bmod < y, p^k >$. Therefore, the lifting returns the correct result.

Algorithm 8.3 (Bivariate polynomial factorization over $\mathbb{Q}(\alpha)$)**.**
Input: a square-free polynomial $f \in \mathbb{Q}(\alpha)[x, y]$ which is primitive with respect to x and y

Output: the irreducible decomposition $\mathrm{Irr}(f)$

Instructions:

1. choose $a \in \mathbb{Z}$ at random such that $f(x, a)$ remains square-free and $\deg(f(x, a)) = \deg_x(f)$
2. compute $l = \mathrm{Irr}(f(x, a))$ and remove $f_1 = \mathrm{lc}(f(x, a))$ from it
3. set $f = f/\mathrm{lc}(f(x, a))$
4. $f(x, y) = f(x, y + a)$
5. **for** all $f_i \in \mathrm{Irr}(f(x, a))$ compute the smallest $g \in \mathbb{Z}$ such that $gf_i \in \mathbb{Z}[\alpha][x]$ and set $f_i = gf_i$
6. compute the smallest $g \in \mathbb{Z}$ such that
   - $gf(x, a) = \mathrm{lc}(gf(x, a)) \prod_{f_i \in \mathrm{Irr}(f(x,a))} f_i$
   - $gf \in \mathbb{Z}[\alpha][x][y]$

   and set $f = gf$
7. compute a bound B on the coefficients of a factor of f as described in 6.2
8. choose $k > 0$ and a prime p such that $p \nmid \mathrm{lc}_x(f)(0) \cdot \mathrm{disc}(\mu)$ and $p^k > 2B$
9. **for** $f_i \in l$ set $\tilde{f_i} = \mathrm{lc}(f_i)^{-1} f_i \bmod p^k$

10. $\tilde{f} = (\mathrm{lc}_x(f)(0))^{-1} f \bmod p^k$
11. **call** algorithm 6.5 with input $\tilde{f}$, $\tilde{f}_i$, p, k, and $n = \deg_y(f) + 1$ to compute $f_i^{(n)}$ such that $\tilde{f} = \mathrm{lc}_x(\tilde{f}) \prod_i f_i^{(n)} \bmod < y^{n+1}, p^k >$ and $f_i^{(n)}(x, 0) = \tilde{f}_i$
12. compute $D = \mathrm{disc}(\mu)$ and $d_f = \mathrm{res}(\mu, \mathrm{lc}(f))$
13. **call** algorithm 7.1 with input f, $f_i^{(n)}$, p, k, and $d_f D$ to compute the irreducible factors $g_1, \ldots, g_s$ of f
14. $g_j = g_j(x, y - a)$ **for** $j = 1, \ldots, s$
15. **return** $g_1, \ldots, g_s$

Note that $\mathrm{lc}_x(f)(0) \in \mathbb{Z}$ holds in step 8 due to steps 3 and 6. Therefore, steps 9 and 10 cannot fail and $\tilde{f} = \mathrm{lc}_x(\tilde{f}) \prod \tilde{f}_i \bmod < y, p^k >$ holds before step 11. Now step 11 may fail if solving 6.1.2 modulo p failed. As there are only finitely many primes that lead to failure, step 11 eventually returns the correct result.

Chapter 9

Multivariate factorization

We state the multivariate polynomial factorization algorithm to complete the description.

As for the bivariate factorization algorithm, we assume F to be square-free and primitive with respect to any variable occurring in F. Furthermore, we assumed $\frac{\partial F}{\partial x_1} \neq 0$. As F is square-free this is no restriction.

Algorithm 9.1 (Multivariate polynomial factorization over a finite field)**.**
Input: a square-free polynomial $F \in \mathbb{F}_q[x_1, x_2, x_3, \ldots, x_n]$ which is primitive with respect to x_i for $i = 1, \ldots, n$ and $\frac{\partial F}{\partial x_1} \neq 0$

Output: the irreducible decomposition $\mathrm{Irr}(F)$ or "failure"

Instructions:

1. **if** $\mathrm{tdeg}(\gcd(\frac{\partial F}{\partial x_1}, F)) > 0$
   (a) **call** the present algorithm with input $G = \gcd(\frac{\partial F}{\partial x_1}, F)$ and F/G
   (b) **return** $\mathrm{Irr}(F/G) \cup \mathrm{Irr}(G)$
2. choose an evaluation point $a = (a_2, a_3, \ldots, a_n) \in \mathbb{F}_q^{n-1}$ such that:
   - $f = F(x_1, x_2, a_3, \ldots, a_n)$ is square-free
   - $f(x_1, a_2)$ is square-free
   - $\deg_{x_1}(F) = \deg_{x_1}(f) = \deg(f(x_1, a_2))$
   - f is primitive with respect to x_1
   - $\deg_{x_i}(F) = \deg_{x_i}(F(x_1, x_2, \ldots, x_i, a_{i+1}, \ldots, a_n))$

3. **if** there is no such a, pass to a suitable field extension $\mathbb{F}_{q^k}$ and **goto** step 2
4. **call** algorithm 8.1 with input f to obtain the irreducible bivariate factors $f_1, \ldots, f_r$ of f
5. **call** algorithm 6.11 with input F, $f_1, \ldots, f_r$, and $(a_3, \ldots, a_n)$ to obtain m and $p_1, \ldots, p_r$
6. **if** m is non-constant, set $F = m^{r-1}F$, $f_i = m(x_2, a_3, \ldots, a_n) \cdot f_i$ **for** $i = 1, \ldots, r$, and $p_i = mp_i$ **for** $i = 1, \ldots, r$
7. **call** algorithm 6.10 with input F, $f_1, \ldots, f_r$ and $p_1, \ldots, p_r$
8. **if** algorithm 6.10 returns $F_1, \ldots, F_r$ **return** $\mathrm{pp}_{x_1}(F_1), \ldots, \mathrm{pp}_{x_1}(F_r)$
9. $F = F(x_1, x_2 + a_2, \ldots, x_n + a_n)$
   $p_i = p_i(x_2 + a_2, \ldots, x_n + a_n)$ **for** $i = 1, \ldots, r$
   $f_i = f_i(x_1, x_2 + a_2)$ **for** $i = 1, \ldots, r$
10. **call** algorithm 6.8 with input F, $f_1, \ldots, f_r$, $p_1, \ldots, p_r$, and $\deg_{x_3}(F) + 1, \ldots, \deg_{x_n}(F) + 1$ to obtain $F_1, \ldots, F_r$
11. **if** $\mathrm{pp}_{x_1}(F_i) \mid F$ **for** all $i = 1, \ldots, r$
    (a) $F_i = (\mathrm{pp}_{x_1}(F_i))(x_1, x_2 - a_2, \ldots, x_n - a_n)$ **for** $i = 1, \ldots, r$
    (b) **return** $F_1, \ldots, F_r$
12. **return** "failure"

Proof: To show the correctness of the first 3 steps of the above algorithm, the proof of the bivariate case carries over to the multivariate case. That is, since after step 1 $\gcd(\frac{\partial F}{\partial x_1}, F) = 1$, the resultant $\mathrm{res}_{x_1}(F, \frac{\partial F}{\partial x_1})$ is a non-zero polynomial, and thus has only finitely many roots. Furthermore, any root of $\mathrm{lc}_{x_1}(F)$ is a root of $\mathrm{res}_{x_1}(F, \frac{\partial F}{\partial x_1})$. In addition to the bivariate case, $\mathrm{lc}_{x_i}(F)$ has finitely many roots. Let $F = \sum_{i=1}^{d_x} g_i x^i$ and since F is primitive with respect to x only finitely many evaluation points yield a non-constant $\gcd(g_1(a_3, \ldots, a_n), \ldots, g_{d_x}(a_3, \ldots, a_n))$. In total, this show that a large enough field extension eventually contains a suitable evaluation point a. "failure" is returned if and only if there is no one-to-one correspondence between bivariate and multivariate factors, since otherwise algorithm 6.11 had returned the right leading coefficients and algorithm 6.8 had returned the right factors.

To obtain the factorization even in cases where the above algorithm returns "failure", one can make the f_i monic in x_1 by inverting their leading coefficients mod ${x_2}^{\deg_{x_2}(F)+1}$, add the leading coefficient $\mathrm{lc}_{x_1}(F)$ as factor, and use a generalization of algorithm 6.1 to keep the lifted factors monic. Then one has to recombine the lifted factors similar as in 7.1. This, however, is very inefficient.

In algorithm 9.1 we have chosen x_1 as main variable and x_2 as second variable to keep the presentation simple. In our implementation we choose the main variable $x \in \{x_1, \ldots, x_n\}$ such that $\mathrm{lc}_x(F)$ has as few terms as possible to keep down the cost of algorithm 6.11. Besides, in case a non-constant m is returned by algorithm 6.11 this choice contains the term growth in step 6. Furthermore, we try to order the remaining variables decreasingly by $\deg_{x_i}(F)$ to minimize the cost for Hensel lifting. This may not always be possible because we also try to choose the second variable such that the conditions for algorithm 6.11 are satisfied.

On the other hand, the main and second variable x respectively y may be chosen such that $\deg_x(F)$ and $\deg_y(F)$ is minimal to decrease the cost for bivariate factorization. Another possibility is to choose x such that $\mathrm{lc}_x(F)$ has as much terms as possible. This makes the sparse heuristic Hensel lifting more likely to succeed as more starting solutions are known.

Algorithm 9.2 (Multivariate polynomial factorization over $\mathbb{Q}$ or $\mathbb{Q}(\alpha)$)**.**
Input: a square-free polynomial $F \in \mathbb{K}[x_1, x_2, x_3, \ldots, x_n]$ - where $\mathbb{K}$ is either $\mathbb{Q}$ or $\mathbb{Q}(\alpha)$ - which is primitive with respect to x_i for $i = 1, \ldots, n$

Output: the irreducible decomposition $\mathrm{Irr}(F)$ or "failure"

Instructions:

1. **if** $\mathbb{K} = \mathbb{Q}$, compute the lowest common denominator g of the coefficients of F and set $F = gF$
2. **else** compute the smallest $g \in \mathbb{Z}$ such that $gF \in \mathbb{Z}[\alpha][x_1, \ldots, x_n]$ and set $F = gF$
3. choose an evaluation point $a = (a_2, a_3, \ldots, a_n) \in \mathbb{K}^{n-1}$ such that:
   - $f = F(x_1, x_2, a_3, \ldots, a_n)$ is square-free
   - $f(x_1, a_2)$ is square-free

- $\deg_{x_1}(F) = \deg_{x_1}(f) = \deg(f(x_1, a_2))$
- f is primitive with respect to x_1
- $\deg_{x_i}(F) = \deg_{x_i}(F(x_1, x_2, \ldots, x_i, a_{i+1}, \ldots, a_n))$

4. **call** algorithm 8.2 or 8.3 with input f to obtain the irreducible bivariate factors $f_1, \ldots, f_r$ of f
5. **call** algorithm 6.11 with input F, $f_1, \ldots, f_r$, and $(a_3, \ldots, a_n)$ to obtain m and $p_1, \ldots, p_r$
6. **if** m is non-constant, set $F = m^{r-1}F$, $f_i = m(x_2, a_3, \ldots, a_n) \cdot f_i$ **for** $i = 1, \ldots, r$, and $p_i = mp_i$ **for** $i = 1, \ldots, r$
7. **call** algorithm 6.10 with input F, $f_1, \ldots, f_r$ and $p_1, \ldots, p_r$
8. **if** algorithm 6.10 returns $F_1, \ldots, F_r$ **return** $\mathrm{pp}_{x_1}(F_1), \ldots, \mathrm{pp}_{x_1}(F_r)$
9. $F = F(x_1, x_2 + a_2, \ldots, x_n + a_n)$
   $p_i = p_i(x_2 + a_2, \ldots, x_n + a_n)$ **for** $i = 1, \ldots, r$
   $f_i = f_i(x_1, x_2 + a_2)$ **for** $i = 1, \ldots, r$
10. **call** algorithm 6.8 with input F, $f_1, \ldots, f_r$, $p_1, \ldots, p_r$, and $\deg_{x_3}(F) + 1, \ldots, \deg_{x_n}(F) + 1$ to obtain $F_1, \ldots, F_r$
11. **if** $\mathrm{pp}_{x_1}(F_i) \mid F$ **for** all $i = 1, \ldots, r$
    (a) $F_i = (\mathrm{pp}_{x_1}(F_i))(x_1, x_2 - a_2, \ldots, x_n - a_n)$ **for** $i = 1, \ldots, r$
    (b) **return** $F_1, \ldots, F_r$
12. **return** "failure"

Chapter 10

Newton Polygon Methods

As we have seen, Newton polygons carry powerful information about the irreducible decomposition of a bivariate polynomial. Since Newton polygons only depend on the support of a polynomial, they are independent of the underlying coefficients.

The Newton polygon of a bivariate polynomial f can be computed by means of Graham scan [Gra72], which takes $\mathcal{O}(m \log(m))$, where m denotes the number of non-zero terms of f.

A classic theorem by Ostrowski [Ost99] relates the decomposition of a polynomial to the decomposition of its Newton polytope as Minkowski sum:

Let $f, g, h \in R[x_1, \ldots, x_n]$ such that $f = gh$ then $\mathcal{N}(f) = \mathcal{N}(g) + \mathcal{N}(h)$, where $\mathcal{N}(f)$ denotes the Newton polytope of f.

Now if one can show that the Newton polytope of a polynomial f is integrally indecomposable, then f is irreducible. Since the Newton polytope is independent of the ring R, f is irreducible over any ring S that contains R. Therefore, if R is a field, f is absolutely irreducible over R.

For the case of bivariate polynomials Gao and Lauder in [GL01] show that deciding whether an integral convex polygon is integrally indecomposable is NP-complete.

However, some very simple irreducibility tests can be derived:
Let f be a two term bivariate polynomial in the indeterminates x, y of degree n in x and m in y. Then f is absolutely irreducible if $\gcd(n, m) = 1$ [Gao01, Corollary 4.4].

Let $f = ax^n + by^m + cx^u y^v + \sum_{(i,j)} c_{ij} x^i y^j$ with non-zero a, b, c such that the Newton polygon of f is of the form $(n, 0), (0, m), (u, v)$ and $\gcd(n, m, u, v) = 1$ then f is absolutely irreducible [Gao01, Example 1].

All algorithms for computing the irreducible decomposition of a bivariate polynomial f described so far depend on the dense size of the input. That is, their run-time depends on $\deg_x(f)$ and $\deg_y(f)$. Let the convex size of f be the number of integral points contained in its Newton polygon $\mathcal{N}(f)$, and its sparse size be the number of non-zero terms. Furthermore, the dense size of f is defined as $(\deg_x(f) + 1)(\deg_y(f) + 1)$. If we assume f to be primitive with respect to x and y, then its Newton polygon $\mathcal{N}(f)$ is always contained in $[0, \deg_x(f)] \times \big[0, \deg_y(f)\big]$; and it contains at least one point on the x-axis and one point on the y-axis. Now one can apply an algorithm described in [BL12, Algorithm 1] to it that only uses the following elementary operations:

$$\rho : (i, j) \to (i - j, j)$$

and

$$\sigma : (i, j) \to (j, i)$$

and

$$\tau_k : (i, j) \to (i + k, j).$$

Algorithm 1 of [BL12] returns a map U

$$U : (i, j) \to M \begin{pmatrix} i \\ j \end{pmatrix} + \begin{pmatrix} a \\ b \end{pmatrix},$$

where $M \in GL(\mathbb{Z}, 2)$, $a, b \in \mathbb{Z}$ that when applied to $\mathcal{N}(f)$ yields a polygon whose dense size is at most 9π, where π denotes the convex size of $\mathcal{N}(f)$ (see [BL12, Theorem 1.2]).

If U is applied to the monomials of a polynomial f, then $U(f)$ is irreducible if and only if f is irreducible. One can recover the factors of f from those of $U(f)$ by applying U^{-1} and normalizing. In 11.3.2 we give an example that illustrates the power of the reduction of convex-dense to dense polynomials.

Based on Ostrowski's theorem, [ASGL04] derive a Las Vegas algorithm to compute the irreducible factorization of a bivariate polynomial f given a decomposition of $\mathcal{N}(f)$ which turns out to be very efficient over $\mathbb{F}_2$, see

[Sal08]. However, computing the integral decomposition of an integral polygon is NP complete [GL01, Proposition 14]. Nevertheless, they devise a pseudo-polynomial time algorithm that is able to compute the integral decomposition of an integral polytope.

Chapter 11

Implementation and results

11.1 `factory`

`factory` is a `C++` library originally developed by R. Stobbe, J. Schmidt, H. Schönemann, G. Pfister and G.-M. Greuel at the University of Kaiserslautern to support operations such as greatest common divisor, factorization, and resultants. `factory` is used by `Singular` [DGPS12] for these very operations.

The following coefficient domains are supported by `factory`: $\mathbb{Z}, \mathbb{Q}, \mathbb{Q}(\alpha), \mathbb{F}_p, \mathbb{F}_q$, where two different representations of extensions of $\mathbb{F}_p$ are available: either by roots of Conway polynomials [JLPW95] with addition tables for $p \leq 251$ and $p^n < 2^{16}$ or as residue classes modulo a given irreducible polynomial. For finite fields p must be less than 2^{29}. Polynomial data is implemented in sparse recursive representation as linked list, that is, *every* polynomial is considered univariate with coefficients being, possibly, polynomial again. As interface the type `CanonicalForm` is used, which can represent polynomial data as well as coefficients. To distinguish between these two, each variable has a level which is an integer. Coefficients of the intrinsic domains $\mathbb{Z}, \mathbb{Q}, \mathbb{F}_p$ have level 0. Algebraic variables used to represent $\mathbb{F}_q, \mathbb{Q}(\alpha)$ have level < 0, whereas polynomial variables have level > 0. Hence, each polynomial has a level, namely the maximum of the levels of its variables.

As sparse recursive representations are rather ill-suited for univariate operations, there is an interface to Shoup's `NTL` library [Sho] to handle univariate operations such as factorization and gcd over $\mathbb{F}_p, \mathbb{F}_q$, and $\mathbb{Z}$. The author has also implemented an interface to `FLINT` [HPN$^+$] since in contrast

to NTL, FLINT is actively developed, thread-safe, and supports univariate polynomials over $\mathbb{Q}$. Besides this FLINT provides almost the same functionality as NTL at a comparable or even better level of performance.

During the course of this thesis the author has revised and reimplemented large parts of factory. This was necessary since many implemented algorithms were outdated and/or badly implemented. Together with the newly implemented algorithms, the author has contributed approximately 30,000 physical lines of code to factory.

11.2 Some implementation details

As shown in section 6.1, lifting univariate polynomials to bivariate polynomials can be implemented such that only univariate polynomial arithmetic and cheap additions of bivariate polynomials are needed. To speed up this step, we use FLINT or NTL to handle the univariate polynomial arithmetic as they surpass factory's built-in polynomial arithmetic in the univariate case. NTL, as well as FLINT, have implemented asymptotically fast univariate arithmetic for various rings. In both, univariate polynomials are represented as dense polynomials, in contrast to factory's sparse linked list representation.

Bivariate polynomial multiplication can be reduced to univariate polynomial multiplication by Kronecker substitution in the following way: Let $f = \sum_{i=0}^{m} f_i y^i$ and $g = \sum_{i=0}^{l} g_i y^i$ be polynomials in $R[x][y]$ with $\deg_x(f_i) \leq d$ and $\deg_x(g_i) \leq d$. Then the product $h = fg$ can be obtained by

$$h(x, x^{2d-1}) = f(x, x^{2d-1})g(x, x^{2d-1})$$

in $\mathcal{O}(M(ld))$ operations over R if we assume $l > m$ (see [vzGG03]).

In [Har07] faster versions of Kronecker substitution are described, using 2 or 4 evaluation points. We have implemented the reciprocal evaluation point method which uses two evaluation points.

One frequently used operation is multiplication in $R[x][y]/(y^n)$. Here only the first $(2d-1)(n-1)$ coefficients of the above product are needed. FLINT and NTL provide special functions to compute only the first m coefficients of

a product which we use in conjunction with the above methods.

Another way to speed up multiplication in $R[x][y]/(y^n)$ is Karatsuba multiplication.
Let us assume $m, l \leq n$, then

$$\begin{aligned} f =& f_0 + f_1 y^{\frac{n}{2}} \\ g =& g_0 + g_1 y^{\frac{n}{2}} \end{aligned}$$

where $\deg(f_i), \deg(g_i) \leq \frac{n}{2}$. Hence,

$$h = f \cdot g = f_0 g_0 + (f_0 g_1 + f_1 g_0) y^{\frac{n}{2}}$$

in $R[x][y]/(y^n)$. One can proceed similarly in the recursive calls to compute $f_0 g_1$ and $f_1 g_0$ in $R[x][y]/(y^{\frac{n}{2}})$.

It is easy to generalize this method to the multivariate case $R[x_1, \ldots, x_n]/(x_2^{k_2}, \ldots, x_n^{k_n})$ by recursion on the variables. We found Kronecker substitution to be superior to Karatsuba multiplication if asymptotically fast multiplication is available. However, this is not always the case, for instance, if a finite field is represented by roots of a Conway polynomial.

In the polynomial time recombination algorithm it is necessary to compute linearized logarithmic derivatives of the form

$$f/g \cdot g' \bmod y^k$$

Since g is monic in x, it is possible to use Newton iteration (see [vzGG03, Chapter 9]) to compute an inverse of the reversal $\mathrm{rev}_l(g) \bmod (x^{m-l+1}, y^k)$. Hence, computing f/g comes down to one computation of an inverse of $\mathrm{rev}_l(g) \bmod (x^{m-l+1}, y^k)$ and one multiplication of this inverse with $\mathrm{rev}_m(f) \bmod (x^{m-l+1}, y^k)$.

However, using Newton inversion is best if asymptotically fast multiplication is used. In case this is not possible, one can adapt a method of Burnikel and Ziegler [BZ98] - originally proposed for fast division of integers. This method uses Karatsuba multiplication to compute an approximation to the remainder and corrects this approximation until the actual remainder is recovered.

f/g has to be computed mod y^k for different k. Assume f/g mod y^k was already computed for some k, and one wants to compute f/g mod y^l for $l \geq k$. Then one can write

$$\begin{aligned} f &= f_0 + f_1 y^k \\ g &= g_0 + g_1 y^k \\ q &= f/g = q_0 + q_1 y^k \end{aligned}$$

A short computation gives

$$f - q_0 g = q_1 g y^k \bmod y^l$$

Since this is divisible by y^k, it suffices to compute q_1 as

$$\frac{\frac{f-q_0 g}{y^k}}{g} \bmod y^{l-k}$$

For all these operations one has to keep in mind: The polynomials that occur in Hensel lifting tend to become dense due to shifting the evaluation point and by requiring the lifted factors to be monic in the main variable. Hence, it makes sense to use methods that can manage dense input. Nevertheless, the conversion between `factory` and `NTL` respectively `FLINT` does constitute overhead.

11.3 Benchmarks

All tests in this section were performed on a Xeon X5460 with 64 GB Ram using a development version of `Singular 3-1-6` [1], `FLINT 2.3` and `NTL 5.5.2` compiled with gcc-4.5.4 unless mentioned otherwise. For convenience we drop the version numbers in the following. Timings are given in milliseconds.

11.3.1 A bivariate polynomial over $\mathbb{F}_2$

```
 f = x^4120 + x^4118*y^2 + x^3708*y^400 + x^3706*y^402
+ x^2781*y^1300 + x^2779*y^1302 + x^1339*y^2700
+ x^927*y^3100 + y^4000 + x^7172*y^4167 + x^8349*y^4432
+ x^8347*y^4434 + x^6760*y^4567 + x^5833*y^5467
+ x^5568*y^7132 + x^11401*y^8599
```

[1] available from `https://github.com/Singular/Sources/tree/master`, git revision 08ec4da8348e9b9c3f1b17d21725a8cef13b289a

The above polynomial is taken from [Sal08]. It factorizes as follows:

```
g = x^5568*y^4432 + x^1339 + x^927*y^400 + y^1300
h = x^5833*y^4167 + x^2781 + x^2779*y^2 + y^2700
```

First, f is transformed to

```
ff=y^8120+x^2*y^8116+x^4432*y^7917+x^4434*y^7913+x^400*y^7308+
x^402*y^7304+x^4167*y^7005+x^8599*y^6802+x^4567*y^6193+
x^1300*y^5481+x^1302*y^5477+x^5467*y^4366+x^2700*y^2639+
x^7132*y^2436+x^3100*y^1827+x^4000
```

then ff splits into

```
gg= y^2639+x^4432*y^2436+x^400*y^1827+x^1300
hh= y^5481+x^2*y^5477+x^4167*y^4366+x^2700
```

As one can see, the first factor gg has derivative zero with respect to x. Hence $\gcd(ff, \frac{\partial ff}{\partial x}) = hh$. Now each factor can be proven to be irreducible by applying [Gao01, Example 1].

Singular	Magma V2.19-3
3,680	80,100

Table 11.1: Abu Salem benchmark

We attempted to factorize f with `Maple 16` but aborted the computation after 1,000,000 milliseconds.

11.3.2 A family of sparse bivariate polynomials over $\mathbb{F}_{17}$

This family of sparse bivariate polynomials is given in [Ber99] by

$$f_n = x^n y^n + x^{\lfloor \frac{n}{2} \rfloor + 1} y^{\lfloor \frac{n}{2} \rfloor}(y+1) + x^2 y + (n+1)xy + (n^2+3)x - 2 \in \mathbb{F}_{17}[x,y]$$

In [Ber99] the first 700 of these polynomials are factorized. Due to the algorithm by [BL12] we can factorize the first 70,000 polynomials. Each polynomial is transformed to a polynomial of the form

$$\underbrace{(x^{\lfloor \frac{n}{2} \rfloor + 1} + x + (n^2+3))}_{g_0} \cdot y + \underbrace{x^n + x^{\lfloor \frac{n}{2} \rfloor + 1} + (n+1)x - 2}_{g_1}$$

Now if $\gcd(g_0, g_1) = 1$, the polynomial is irreducible as its degree in y is one. Otherwise, one just needs to factorize $\gcd(g_0, g_1)$ which is a univariate polynomial. Note that this experiment was performed on an i7-860 with 8 GB Ram using `NTL` for the univariate factorizations.

11.3.3 Bivariate Swinnerton-Dyer polynomials

To demonstrate the polynomial time behaviour of the implemented factor recombination algorithm, we consider the following family of polynomials due to Swinnerton-Dyer [Zip93], which are also considered in [BLS+04]:

$$S_k(x,y) = \prod x \pm \sqrt{y+1} \pm \sqrt{y+2} \pm \ldots \pm \sqrt{y+k}$$

S_k has total degree $d = 2^k$ and is irreducible, when considered in $\mathbb{F}_p[x,y]$, if $p > k$; but for any evaluation at y it has 2^{k-1} factors of degree 2 in $\mathbb{F}_p[x]$. However, since evaluating at x exhibits much fewer modular factors, for example, $S_7(4134,y)$ has only 5 factors over $\mathbb{F}_{43051}$, one can consider $S'_k = S_k(x^2,y)S_k(y^2,x)$. For odd k, these polynomials are monic of degree 2^{k+1} and symmetric. Hence, these polynomials do not admit the above shortcut, but one can first substitute x for x^2 and y for y^2, factorize the resulting polynomial, and prove irreducibility of each obtained factor after reverting the variable substitution.

	`Singular`	`Magma V2.19-3`
S'_7	1,540	600
S'_9	69,190	9,070

Table 11.2: Time to factorize Swinnterton-Dyer polynomials over $\mathbb{F}_{43051}$

	univariate factorizations	lifting and recombination
$S'_7(x,y)$	30	890
$S_7(x^2,y)$	90	30
$S_7(y^2,x)$	90	30
$S'_9(x,y)$	580	43,700
$S_9(x^2,y)$	1,300	1,260
$S_9(y^2,x)$	1,300	960

Table 11.3: Time to factorize Swinnterton-Dyer polynomials over $\mathbb{F}_{43051}$ with `Singular`

The timings in the second table give more detailed timing information on our implementation. Note that we computed 3 univariate factorizations per variable. Hence, in total 18 univariate factorizations were computed. S'_9 has

131,839 terms.

We attempted to factorize S_7' with `Maple 16` but aborted the computation after 1,000,000 milliseconds.

11.3.4 Multivariate polynomials over $\mathbb{Q}$

The following benchmark was proposed in [Fat03] and is also considered in [MP11] to demonstrate the gain of using parallelized multiplication in `Maple 14`.

f	multiply $f \cdot (f+1)$	factorize
$(1+x+y+z)^{20}+1$	460	670
$(1+x^2+y^2+z^2)^{20}+1$	490	720
$(1+x+y+z)^{30}+1$	6,230	4,370
$(1+x+y+z+s)^{20}+1$	80,190	22,100

Table 11.4: `Singular`

f	multiply $f \cdot (f+1)$	factorize
$(1+x+y+z)^{20}+1$	280	5,630
$(1+x^2+y^2+z^2)^{20}+1$	310	6,140
$(1+x+y+z)^{30}+1$	3,490	96,760
$(1+x+y+z+s)^{20}+1$	11,680	264,790

Table 11.5: `Magma V2.19-3`

f	multiply $f \cdot (f+1)$	factorize
$(1+x+y+z)^{20}+1$	54	3,851
$(1+x^2+y^2+z^2)^{20}+1$	44	5,168
$(1+x+y+z)^{30}+1$	307	24,516
$(1+x+y+z+s)^{20}+1$	1,209	88,356

Table 11.6: `Maple 16` showing real time

The difference between the time for multiplication and factorization is due to the following: Multiplication is carried out in `Singular`, which uses a sparse distributive representation for polynomials, whereas the factorization takes place in `factory`.

11.3.5 Bivariate polynomials over $\mathbb{Q}(\alpha)$

For this test, random bivariate polynomials over $\mathbb{Q}(\alpha)$, where α is a root of $1000x^8 + x^5 + 233x + 1$, were produced. These polynomials are not square-free, their coefficients are at most 110 bits and 148 bits integers respectively after clearing denominators over $\mathbb{Q}$. The first polynomial has 54 terms and the second has 110 terms.

	`Singular`	`Magma V2.19-3`	`Maple 16`
$f1$	780	38,880	1,511
$f2$	6,640	1,029,690	11,730

Table 11.7: Time to factorize random bivariate polynomials over $\mathbb{Q}(\alpha)$

minimal polynomial	univariate factorizations	Hensel lifting	total time
m_{11}	6,650	1,400	8,100
m_{12}	6,550	2,300	8,930
m_{13}	46,030	10,430	56,630

Table 11.8: Time to factorize example 1 from [BCG10] with `Singular`

minimal polynomial	univariate factorizations	Hensel lifting	total time
m_{21}	277,730	16,640	294,790
m_{22}	355,740	94,740	452,690
m_{23}	2,466,520	399,190	2,875,420

Table 11.9: Time to factorize example 2 from [BCG10] with `Singular`

In table 11.8 and 11.9 we list the times needed to factorize example 1 and example 2 from [BCG10] modulo different minimal polynomials. Note that we have computed two univariate factorizations per variable. Example 1

has total degree 50, height $< 2^{115}$, one factor of total degree 10, and one factor of total degree 40 modulo m_{1i}. Example 2 has total degree 100, height $< 2^{151}$, one factor of total degree 10, and one factor of total degree 90 modulo m_{2i}. For example 1 we have used

$$\begin{aligned}
m_{11} =& 90 - 87a^5 - 13a^4 - 87a^3 - 63a^2 - 31a,\\
m_{12} =& a^5 - 2a^4 + 7503a^3 + 409158a^2 + 17755875a - 5213231199,\\
m_{13} =& -74527024257430706547a^5 + 413358200277336196784 94a^4 -\\
& 88027991001342996205706 28a^3 + 92207339154192303245514 3510a^2 -\\
& 52359505826960725537487720825a +\\
& 10058909037265011125842957786 04
\end{aligned}$$

m_{11} is used in [BCG10] to construct the example, m_{12} is obtained from m_{11} via `Pari`'s [The12] 'polred' function, and m_{13} is computed by the algorithm from [BCG10]. m_{21}, m_{22}, and m_{23} were obtained in the same way:

$$\begin{aligned}
m_{21} =& -31 + 18a^{10} + 16a^9 - 22a^8 + 38a^7 - 48a^6 - 87a^4 - 13a^3 - 87a^2 - 63a,\\
m_{22} =& a^{10} - 5a^9 - 7557a^8 - 234341a^7 + 15536729a^6 + 1443526148a^5 +\\
& 50269318270a^4 + 956924873772a^3 + 9666707037426a^2 + 33611122836488a -\\
& 116472457328530,\\
m_{23} =& -433811768034816a^{10} + 336536057422207457 28a^9 -\\
& 41979440162252823423552a^8 + 89833163562701240347828 80a^7 -\\
& 83044540410217746770845864 8a^6 + 38782444401126273604523137248a^5 -\\
& 14474636361201530380502474338 13a^4 +\\
& 10975727688393269142516109555 4960a^3 -\\
& 64132001891772527348773447093 81706a^2 +\\
& 16825349765020561055047524546 8397336a -\\
& 18814326168928741116766900161 71428193
\end{aligned}$$

`Maple 16` needed 5,073,001 ms to factorize example 1 modulo m_{11}, 5,057,951 ms modulo m_{12}, and 13,446,070 ms modulo m_{13}. We have also tried to factorize example 2 modulo m_{21}, m_{22}, and m_{23} via `Maple 16`, but had to abort the computation as `Maple 16` used all of the available main memory. `Magma V2.19-3` needed 15,042,240 ms to factorize example 1 modulo m_{11}, 14,047,390 ms modulo m_{12}, and 33,135,530 ms modulo m_{13}. We aborted all attempts to factorize example 2 via `Magma V2.19-3` modulo m_{2i} after one day of computation.

11.3.6 Very sparse polynomials over $\mathbb{Q}$

The following polynomials are from [Ina05]. They were used to illustrate the expression swell caused by shifting the evaluation point to zero for Hensel lifting.

$$\begin{aligned} f_k =&(x^2y^2z + x(y^k + z^k) + 3y + 3z - 3z^2 - 2y^{k/2}z^{k/2}) \\ &(x^3y^2z^2 + x(y^k + z^k) - 2y - 5z + 4y^2 + 3y^{k/2}z^{k/2}) \end{aligned}$$

and

$$\begin{aligned} g_k =&((x(y^3 + 2z^3) + 5yz)(x(y + 4z) + 2) + (2x - 7)(y^kz^k - y^{k-1}z^{k-1})) \\ &((x(3y^3 + 4z^3) + 3yz)(x(y + 3z) + 7) - (3x + 5)(y^kz^k - y^{k-1}z^{k-1})) \end{aligned}$$

k	`Singular`	`Magma V2.19-3`	`Maple 16`
10	10	10	90
20	50	0	160
30	80	20	369
40	180	40	870
50	250	70	2,110

Table 11.10: Time to factorize polynomials f_k

k	Singular	Magma V2.19-3	Maple 16
5	10	20	110
6	20	20	59
7	20	20	70
8	30	30	99
9	30	40	130
10	30	30	139
11	30	50	370
12	30	60	250
13	30	70	250
14	40	70	399
15	30	90	359

Table 11.11: Time to factorize polynomials g_k

11.3.7 Multivariate polynomials over different domains

For these tests we have created 1,000 polynomials as the product of three random polynomials f_1, f_2, and f_3 over $\mathbb{F}_2$, $\mathbb{F}_2[t]/(t^2+t+1)$, $\mathbb{F}_{17}$, $\mathbb{F}_{17}[t]/(t^2+t+1)$, $\mathbb{F}_{43051}$, $\mathbb{F}_{536870909}$, and $\mathbb{Z}$ in 7 variables. In the figures the average computation time of `Singular` is indicated by a blue line. For the specifications of the random polynomials see tables 11.12 and 11.13.

	total degree	random coefficients in	percentage of zero terms
f_1	6	$[1, 100)$	98
f_2	6	$[1, 1000)$	97
f_3	4	$[1, 10000)$	95

Table 11.12: Three sparse random factors

	total degree	random coefficients in	percentage of zero terms
f_1	6	$[1, 100)$	99
f_2	6	$[1, 1000)$	99
f_3	4	$[1, 10000)$	98

Table 11.13: Three very sparse random factors

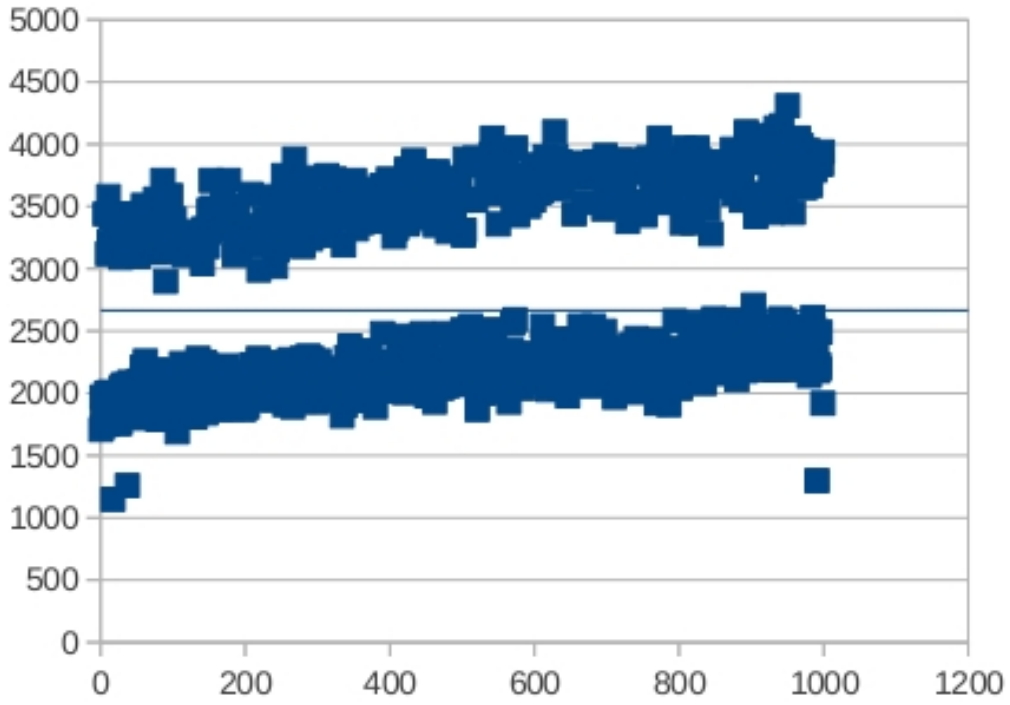

Figure 11.1: Multivariate polynomials in 7 variables with 3 sparse factors over $\mathbb{Q}$

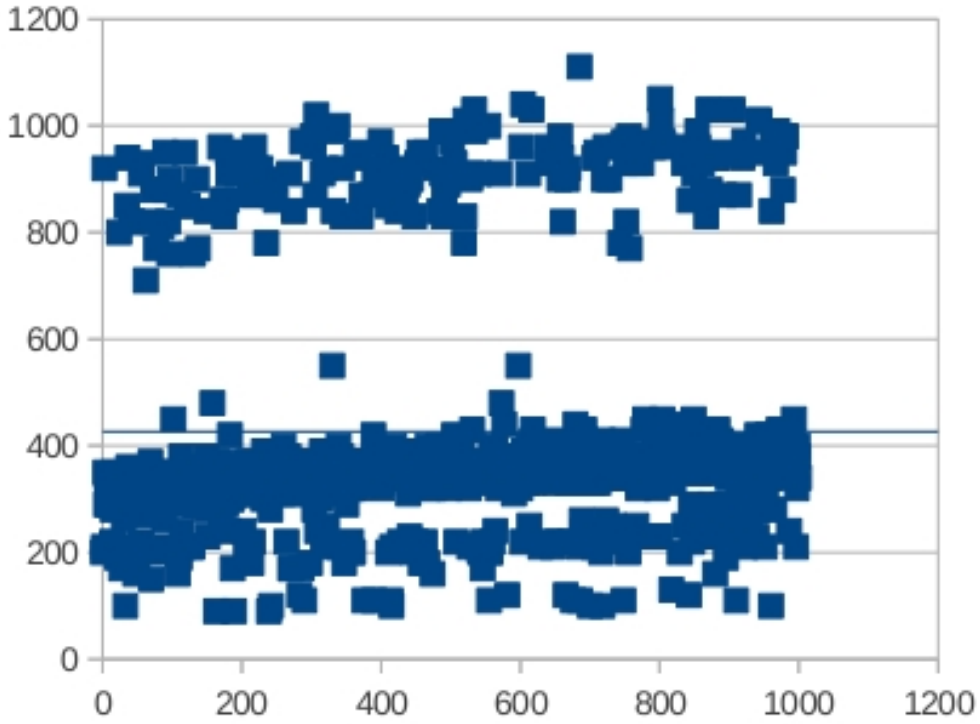

Figure 11.2: Multivariate polynomials in 7 variables with 3 very sparse factors over $\mathbb{Q}$

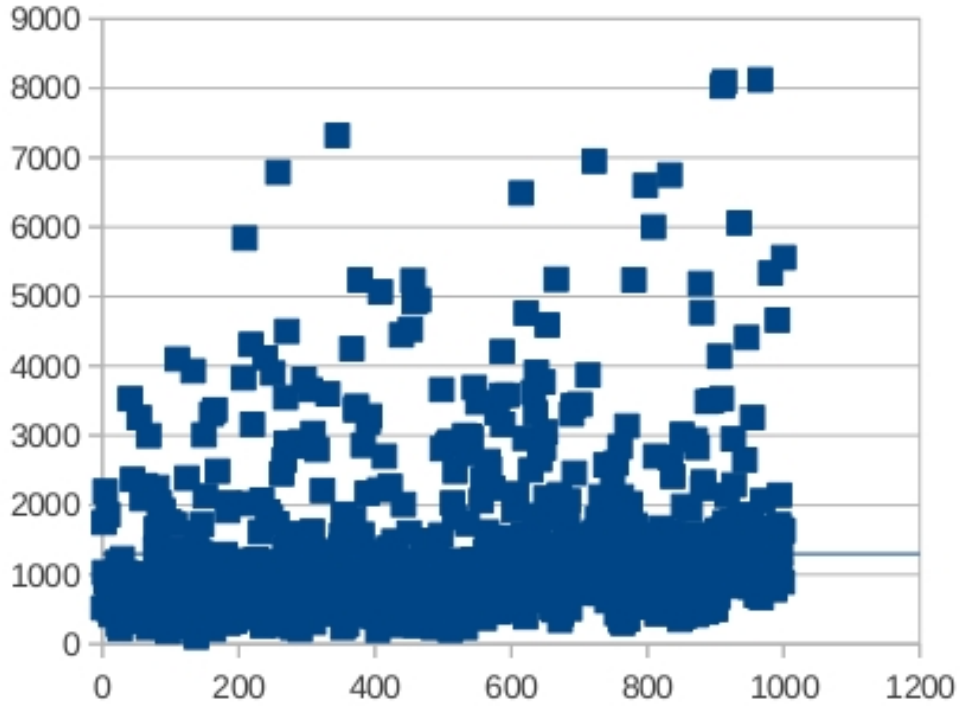

Figure 11.3: Multivariate polynomials in 7 variables with 3 sparse factors over $\mathbb{F}_2$

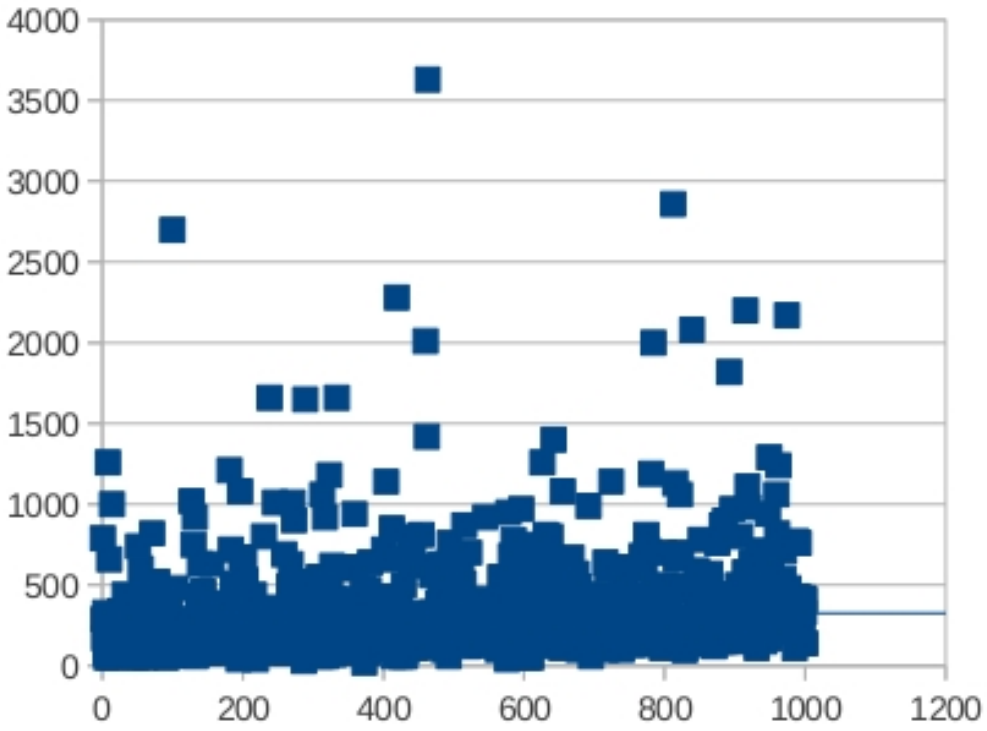

Figure 11.4: Multivariate polynomials in 7 variables with 3 very sparse factors over $\mathbb{F}_2$

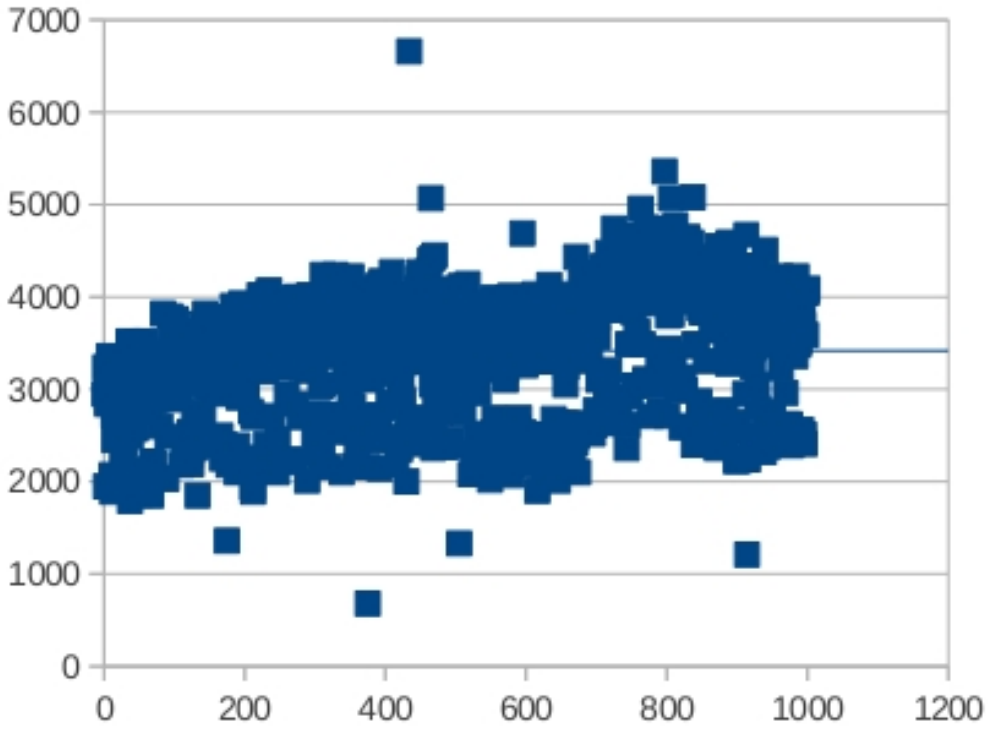

Figure 11.5: Multivariate polynomials in 7 variables with 3 sparse factors over $\mathbb{F}_{17}$

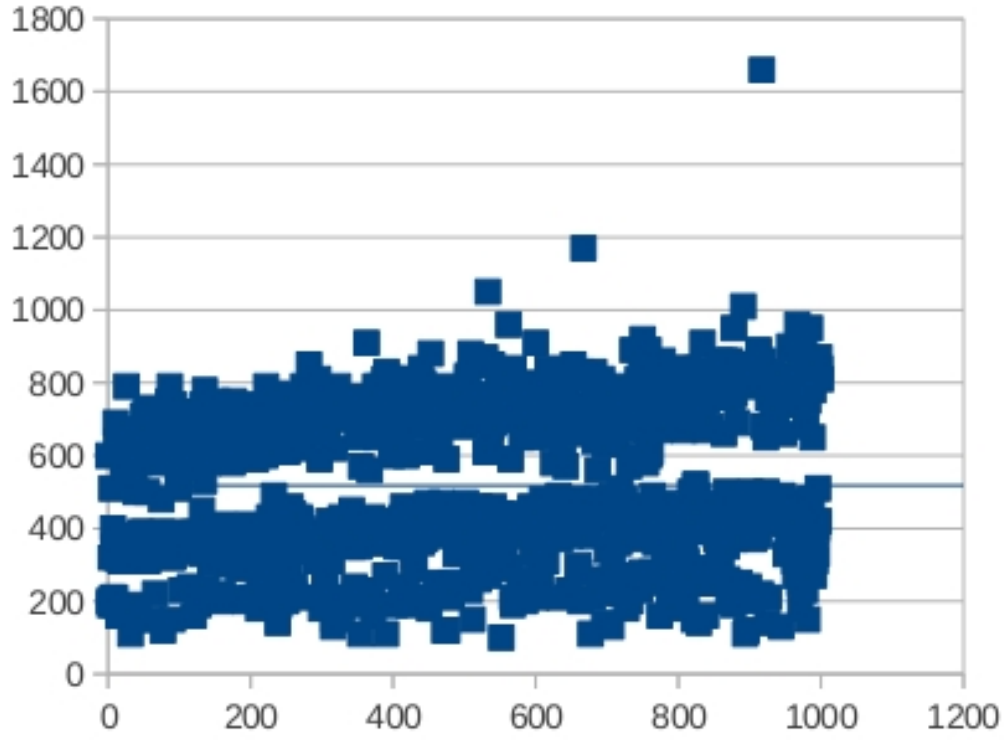

Figure 11.6: Multivariate polynomials in 7 variables with 3 very sparse factors over $\mathbb{F}_{17}$

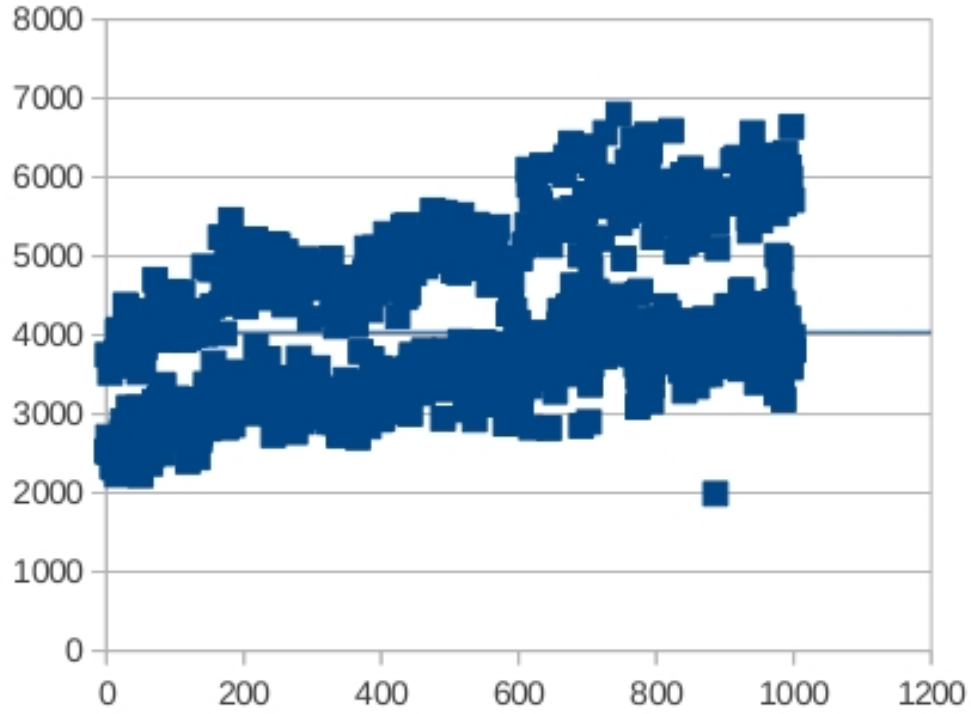

Figure 11.7: Multivariate polynomials in 7 variables with 3 sparse factors over $\mathbb{F}_{43051}$

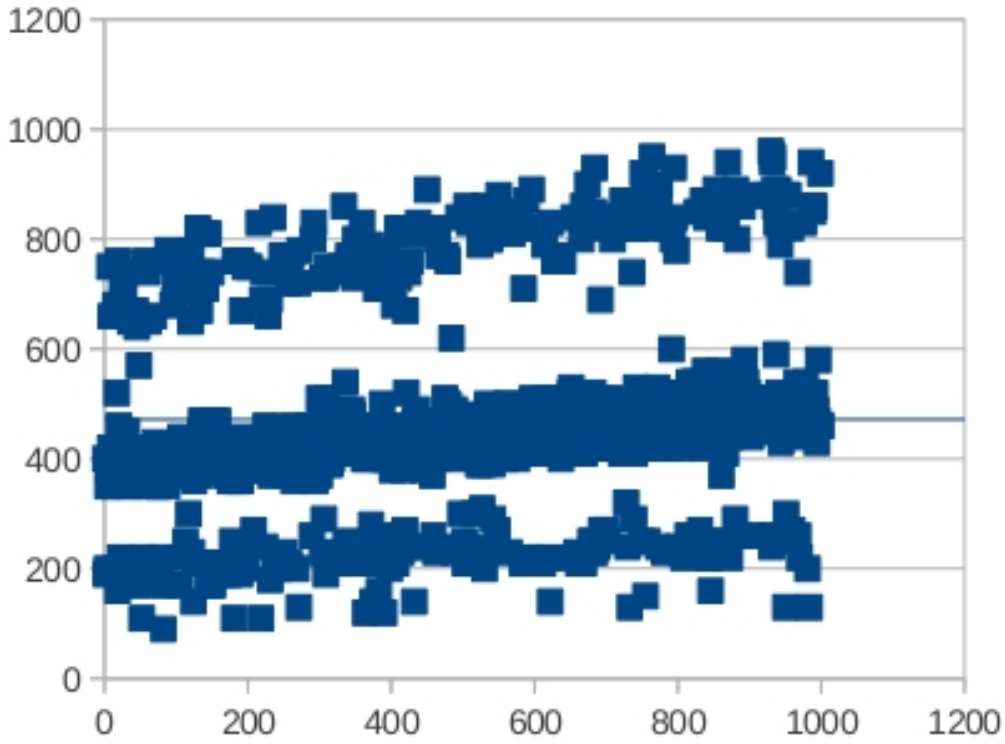

Figure 11.8: Multivariate polynomials in 7 variables with 3 very sparse factors over $\mathbb{F}_{43051}$

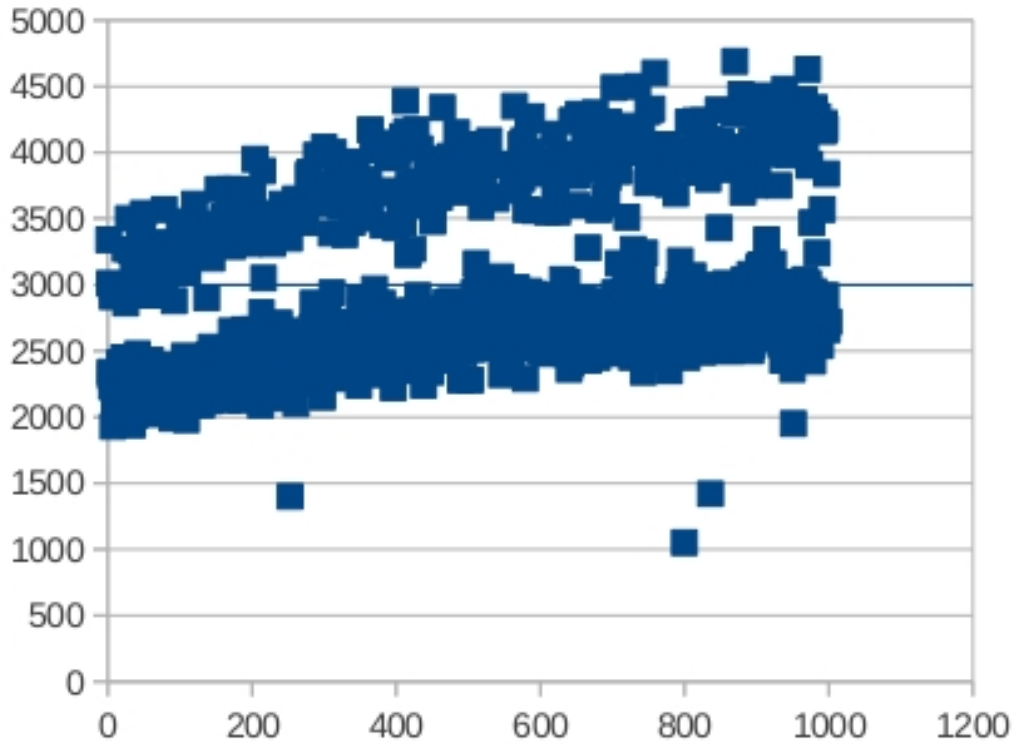

Figure 11.9: Multivariate polynomials in 7 variables with 3 sparse factors over $\mathbb{F}_{536870909}$

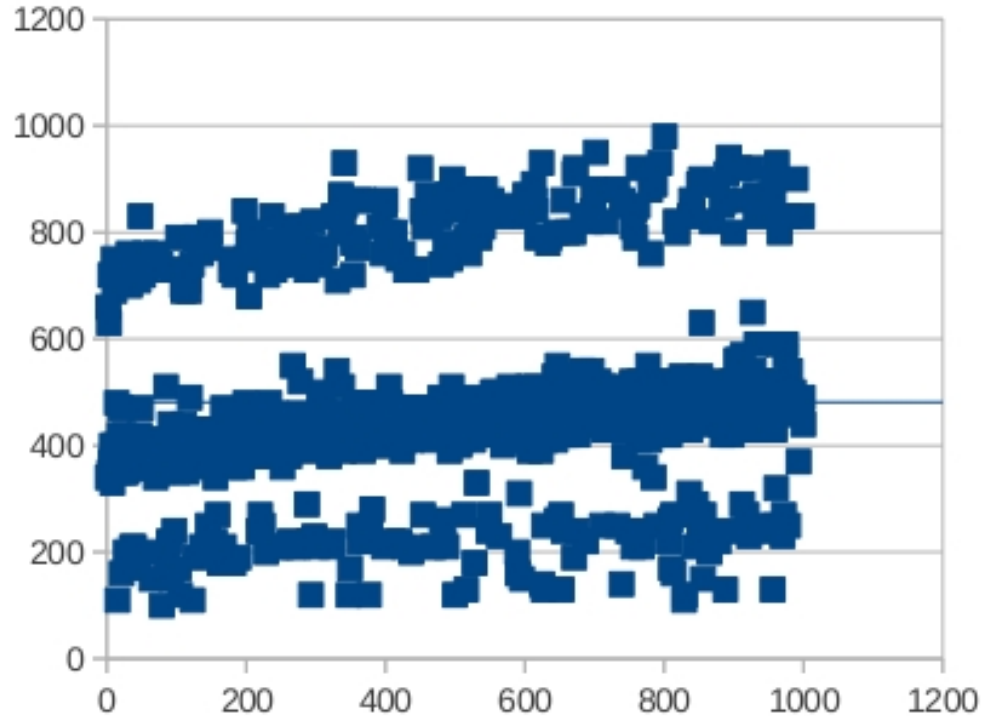

Figure 11.10: Multivariate polynomials in 7 variables with 3 very sparse factors over $\mathbb{F}_{536870909}$

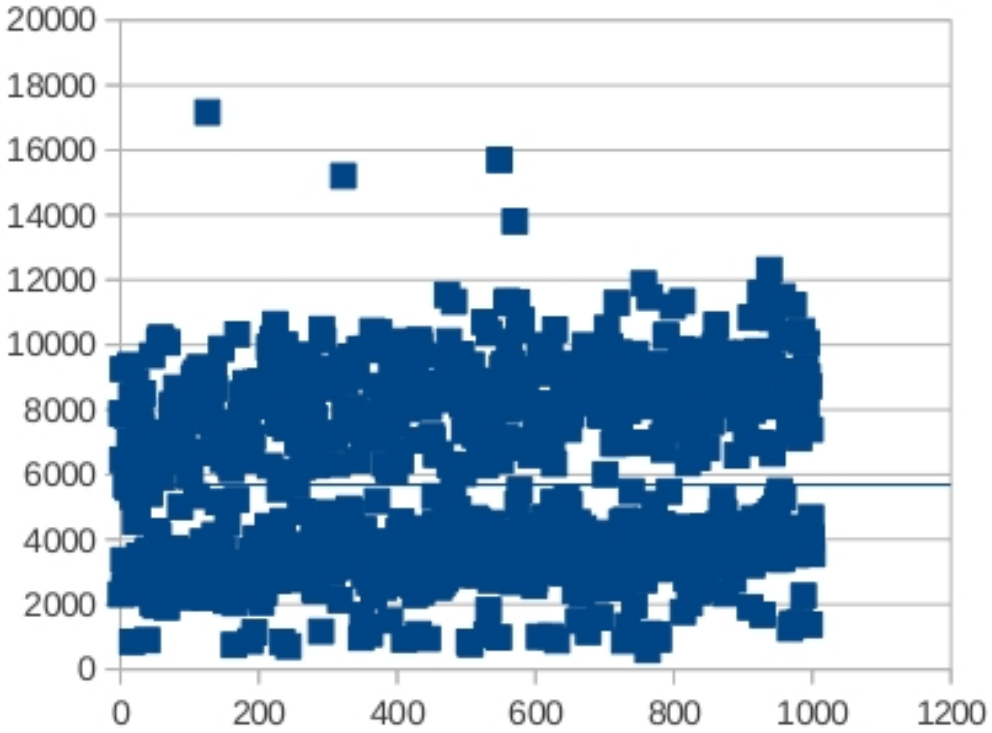

Figure 11.11: Multivariate polynomials in 7 variables with 3 sparse factors over $\mathbb{F}_{2^2}$

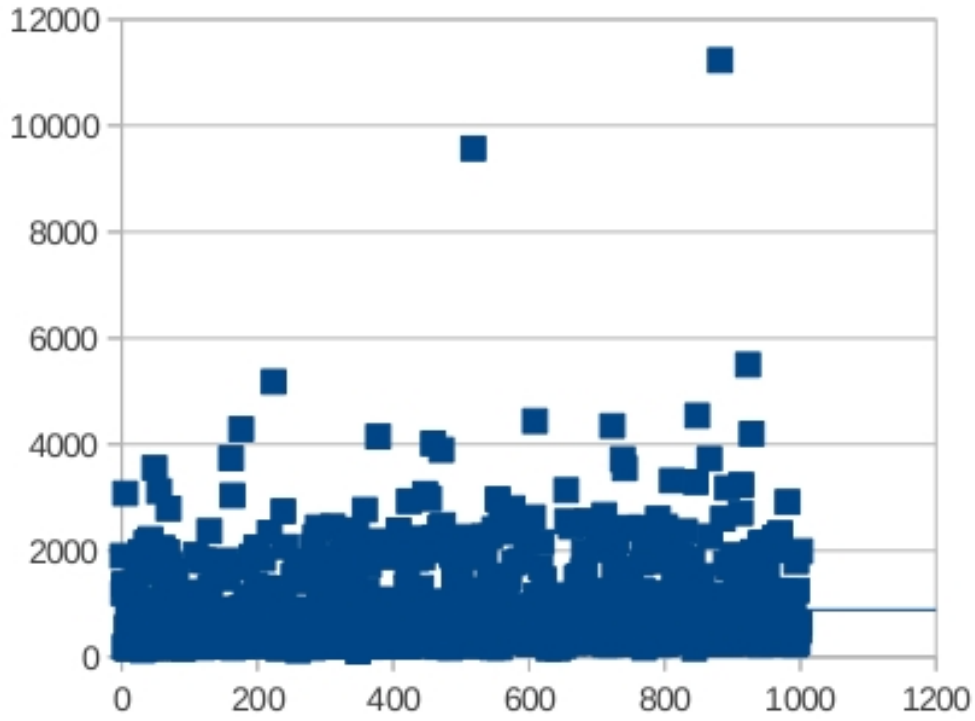

Figure 11.12: Multivariate polynomials in 7 variables with 3 very sparse factors over $\mathbb{F}_{2^2}$

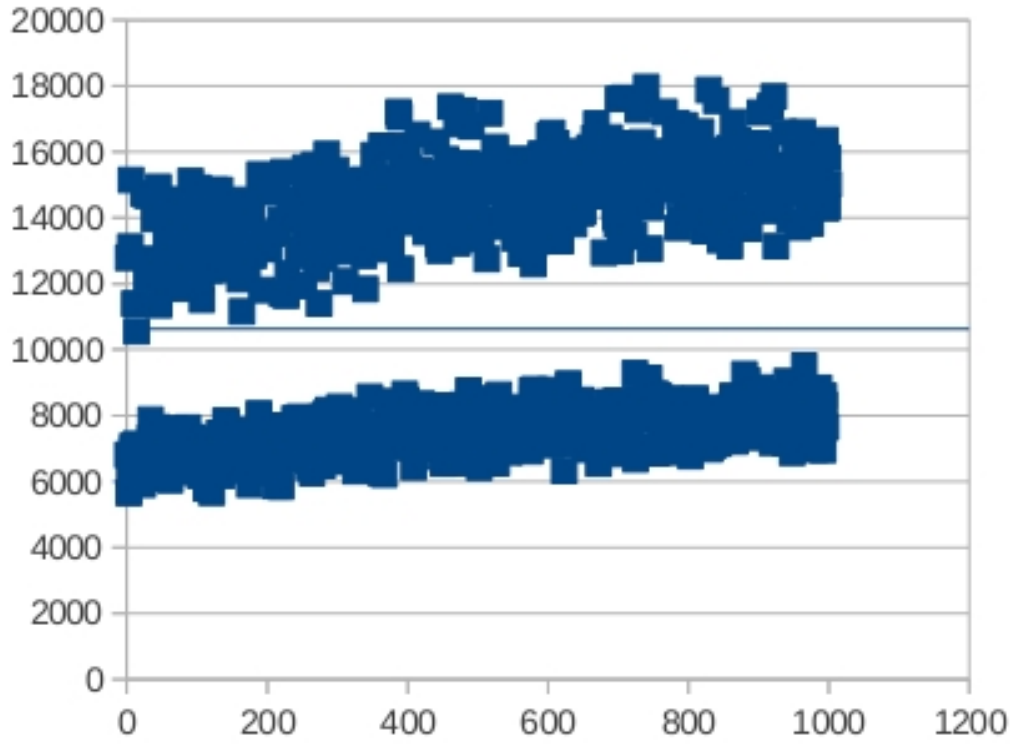

Figure 11.13: Multivariate polynomials in 7 variables with 3 sparse factors over $\mathbb{F}_{17^2}$

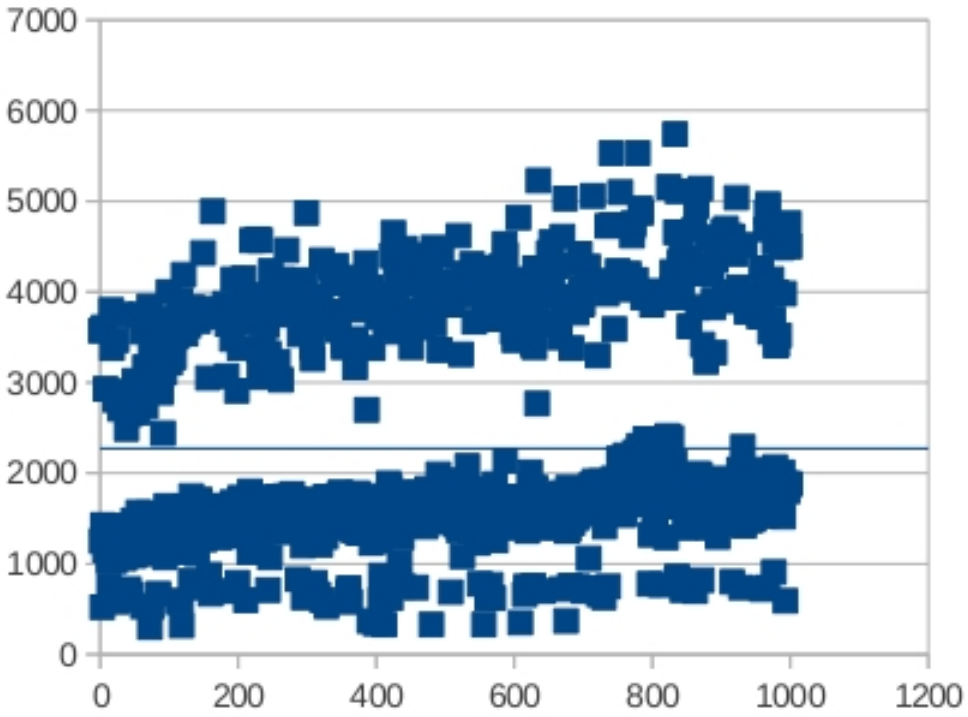

Figure 11.14: Multivariate polynomials in 7 variables with 3 very sparse factors over $\mathbb{F}_{17^2}$

The timings indicate that on average our implementation performs quite well. However, there are some peaks due to failed heuristics and/or poor GCD performance in case the field is small.

11.4 Conclusion and further work

In Chapter 10 we have only presaged the power of factorization algorithms based on Newton polygons. They preserve sparseness as no shifting of the evaluation point is necessary and only depend on the shape of the Newton polygon. A new algorithm by [WCF11], based on ideas by [SS93],[SK99],[Ina05] for Hensel lifting, seems very promising since it is able to factorize bivariate polynomials over $\mathbb{Q}$ of degree $> 100{,}000$. Combining their ideas with the Newton polygon decomposition algorithm of [GL01] may further improve on [ASGL04] in the finite field case.

With the current implementation one may be able to program [Lec10] with little effort which may improve our software in corner cases. With more effort one can also implement [CL07].

On the technical side by interfacing to a thread-safe `FLINT`, we have undertaken first steps towards thread-safety of `factory`. As Hensel lifting usually constitutes the bottleneck of the factorization algorithm, it should be parallelized - [Ber99] shows how to do this. Furthermore, faster parallelized multiplication, as described in [MP11], may show fruitful.

Another improvement is to integrate `FLINT`'s or `NTL`'s polynomial types into `factory`'s `CanonicalForm` type to reduce conversion overhead.

By allowing an arbitrary large characteristic of finite fields, all modular methods may profit as fewer primes need to be used. Besides, it makes the use of Monte-Carlo algorithms possible. One can bound the coefficients occurring in some problem, create several pseudo-primes that exceed the bound, solve the problem modulo these pseudo-primes, and check if the reconstructed result has changed. If this is not the case, one has obtained a result that with high probability is correct. This strategy is used in `NTL` to compute resultants of univariate polynomials over $\mathbb{Z}$.

Instead of the square-free decomposition one should compute the separable decomposition, as pointed out in [Lec08]. This will make step 2 of algorithm 8.1 and step 1 of algorithm 9.1 obsolete.

From the timings one can see that the bottleneck when factoring the examples of [BCG10] is the univariate factorization stage, in contrast to all other examples. One may remedy this by using an algorithm in the spirit of [Rob04] or [Bel04] since they use a modular approach and do not need to compute resultants.

Bibliography

[ASGL04] F. Abu Salem, S. Gao, and A. G. B. Lauder. Factoring polynomials via polytopes. In *Proceedings of the 2004 international symposium on Symbolic and algebraic computation*, ISSAC '04, pages 4–11, New York, NY, USA, 2004. ACM.

[ASZ00] J. Abbott, V. Shoup, and P. Zimmermann. Factorization in z[x]: the searching phase. In *Proceedings of the 2000 international symposium on Symbolic and algebraic computation*, ISSAC '00, pages 1–7, New York, NY, USA, 2000. ACM.

[BCG10] C. Bertone, G. Chèze, and A. Galligo. Modular las vegas algorithms for polynomial absolute factorization. *J. Symb. Comput.*, 45(12):1280–1295, December 2010.

[Bel04] K. Belabas. A relative van Hoeij algorithm over number fields. *J. Symb. Comput.*, 37(5):641–668, 2004.

[Ber99] L. Bernardin. *Factorization of Multivariate Polynomials over Finite Fields*. PhD thesis, ETH Zürich, 1999.

[BL12] J. Berthomieu and G. Lecerf. Reduction of bivariate polynomials from convex-dense to dense, with application to factorizations. *Math. Comp.*, 81(279), 2012.

[BLS+04] A. Bostan, G. Lecerf, B. Salvy, É. Schost, and B. Wiebelt. Complexity issues in bivariate polynomial factorization. In *Proceedings of ISSAC 2004*, pages 42–49. ACM, 2004.

[Bos06] S. Bosch. *Algebra*. Springer-Lehrbuch. Springer, 2006.

[BvHKS09] K. Belabas, M. van Hoeij, J. Klüners, and A. Steel. Factoring polynomials over global fields. *J. Théor. Nombres Bordeaux*, 21(1):15–39, 2009.

[BZ98] C. Burnikel and J. Ziegler. Fast recursive division. Research Report MPI-I-98-1-022, Max-Planck-Institut für Informatik, Im Stadtwald, D-66123 Saarbrücken, Germany, October 1998.

[BZ07] R. P. Brent and P. Zimmermann. A multi-level blocking distinct degree factorization algorithm. *CoRR*, abs/0710.4410, 2007. arXiv:0710.4410v1 [cs.DS].

[CL07] G. Chéze and G. Lecerf. Lifting and recombination techniques for absolute factorization. *Journal of Complexity*, 23(3):380–420, 2007.

[Col71] G. E. Collins. The calculation of multivariate polynomial resultants. *J. ACM*, 18(4):515–532, October 1971.

[DGPS12] W. Decker, G.-M. Greuel, G. Pfister, and H. Schönemann. SINGULAR 3-1-6 — A computer algebra system for polynomial computations. 2012. `http://www.singular.uni-kl.de`.

[dKMW05] J. de Kleine, M. Monagan, and A. Wittkopf. Algorithms for the non-monic case of the sparse modular gcd algorithm. In *Proceedings of the 2005 international symposium on Symbolic and algebraic computation*, ISSAC '05, pages 124–131, New York, NY, USA, 2005. ACM.

[Enc95] M. J. Encarnación. Computing gcds of polynomials over algebraic number fields. *J. Symb. Comput.*, 20(3):299–313, 1995.

[Enc97] M. J. Encarnación. The average number of modular factors in trager's polynomial factorization algorithm. In *Proceedings of the 1997 international symposium on Symbolic and algebraic computation*, ISSAC '97, pages 278–281, New York, NY, USA, 1997. ACM.

[Fat03] R. Fateman. Comparing the speed of programs for sparse polynomial multiplication. *SIGSAM Bull.*, 37(1):4–15, March 2003.

[FR96] P. Fleischmann and P. Roelse. Comparative implementations of berlekamp's and niederreiter's polynomial factorization algorithms. In *Finite Fields and Applications*, pages 73–84. Cambridge University Press, 1996.

[Gao01] S. Gao. Absolute irreducibility of polynomials via newton polytopes. *Journal of Algebra*, 237(2):501 – 520, 2001.

[GCL92] K. O. Geddes, S. R. Czapor, and G. Labahn. *Algorithms for computer algebra*. Kluwer, 1992.

[Gel03] A.O. Gelfond. *Transcendental and Algebraic Numbers*. Dover Phoenix Editions. Dover Publications, 2003.

[GL01] S. Gao and A. G. B. Lauder. Decomposition of polytopes and polynomials. *Discrete & Computational Geometry*, 26:89–104, 2001. 10.1007/s00454-001-0024-0.

[GL02] S. Gao and A. G. B. Lauder. Hensel lifting and bivariate polynomial factorisation over finite fields. *Math. Comput.*, 71(240):1663–1676, October 2002.

[Gra72] R.L. Graham. An efficient algorithm for determining the convex hull of a finite planar set. *Information Processing Letters*, 1(4):132 – 133, 1972.

[Grü67] B. Grünbaum. *Convex polytopes*. Interscience Publ., London, 1967.

[Har07] D. Harvey. Faster polynomial multiplication via multipoint kronecker substitution. *CoRR*, abs/0712.4046, 2007. arXiv:0712.4046v1 [cs.SC].

[HPN+] W. Hart, S. Pancratz, A. Novocin, F. Johansson, and D. Harvey. FLINT: Fast Library for Number Theory. `www.flintlib.org`. Version 2.3.

[HvHN11] W. Hart, M. van Hoeij, and A. Novocin. Practical polynomial factoring in polynomial time. In *Proceedings of the 36th international symposium on Symbolic and algebraic computation*, ISSAC '11, pages 163–170, New York, NY, USA, 2011. ACM.

[Ina05] D. Inaba. Factorization of multivariate polynomials by extended hensel construction. *SIGSAM Bull.*, 39(1):2–14, March 2005.

[JLPW95] C. Jansen, K. Lux, R. Parker, and R. Wilson. *An atlas of Brauer characters*. London Mathematical Society monographs. Clarendon Press, 1995.

[JM09] S. M. M. Javadi and M. B. Monagan. On factorization of multivariate polynomials over algebraic number and function fields. In *Proceedings of the 2009 international symposium on Symbolic and algebraic computation*, ISSAC '09, pages 199–206, New York, NY, USA, 2009. ACM.

[Kal82] E. Kaltofen. Polynomial factorization. In B. Buchberger, G. Collins, and R. Loos, editors, *Computer Algebra*, pages 95–113. Springer Verlag, Heidelberg, Germany, 2 edition, 1982.

[Kal85a] E. Kaltofen. Effective Hilbert irreducibility. *Information and Control*, 66:123–137, 1985.

[Kal85b] E. Kaltofen. Polynomial-time reductions from multivariate to bi- and univariate integral polynomial factorization. *SIAM J. Comput*, 14:469–489, 1985.

[Kal85c] E. Kaltofen. Sparse Hensel lifting. In *EUROCAL 85 European Conf. Comput. Algebra Proc. Vol. 2*, pages 4–17, 1985.

[Kal89] E. Kaltofen. Factorization of polynomials given by straight-line programs. In S. Micali, editor, *Randomness and Computation*, volume 5 of *Advances in Computing Research*, pages 375–412. JAI Press Inc., Greenwhich, Connecticut, 1989.

[Kal90] E. Kaltofen. Polynomial factorization 1982-1986. In D. V. Chudnovsky and R. D. Jenks, editors, *Computers in Mathematics*, volume 125 of *Lecture Notes in Pure and Applied Mathematics*, pages 285–309. Marcel Dekker, Inc., New York, N. Y., 1990.

[Kal92] E. Kaltofen. Polynomial factorization 1987-1991. In I. Simon, editor, *Proc. LATIN '92*, volume 583, pages 294–313, Heidelberg, Germany, 1992. Springer Verlag.

[Kal03] E. Kaltofen. Polynomial factorization: a success story. In *Proceedings of the 2003 international symposium on Symbolic and algebraic computation*, ISSAC '03, pages 3–4, New York, NY, USA, 2003. ACM.

[KL94] E. Kaltofen and A. Lobo. Factoring high-degree polynomials by the black box berlekamp algorithm. In *Proceedings of the international symposium on Symbolic and algebraic*

computation, ISSAC '94, pages 90–98, New York, NY, USA, 1994. ACM.

[KS98] E. Kaltofen and V. Shoup. Subquadratic-time factoring of polynomials over finite fields. *Math. Comp.*, 67(223):1179–1197, 1998.

[KU11] K. S. Kedlaya and C. Umans. Fast polynomial factorization and modular composition. *SIAM J. Comput.*, 40(6):1767–1802, 2011.

[Lec06] G. Lecerf. Sharp precision in hensel lifting for bivariate polynomial factorization. *Math. Comput.*, pages 921–933, 2006.

[Lec08] G. Lecerf. Fast separable factorization and applications. *Applicable Algebra in Engineering, Communication and Computing*, 19(2), 2008. DOI: 10.1007/s00200-008-0062-4. The original publication is available at www.springerlink.com.

[Lec10] G. Lecerf. New recombination algorithms for bivariate polynomial factorization based on hensel lifting. *Applicable Algebra in Engineering, Communication and Computing*, 21:151–176, 2010.

[Lee09] M. M.-D. Lee. Univariate factorization over finite fields. Master's thesis, TU Kaiserslautern, 2009.

[Len83a] A. K. Lenstra. Factoring multivariate polynomials over finite fields. In *Proceedings of the fifteenth annual ACM symposium on Theory of computing*, STOC '83, pages 189–192, New York, NY, USA, 1983. ACM.

[Len83b] A. K. Lenstra. Factoring polynomials over algebraic number fields. In J.A. Hulzen, editor, *Computer Algebra*, volume 162 of *Lecture Notes in Computer Science*, pages 245–254. Springer Berlin Heidelberg, 1983.

[Len84] A. K. Lenstra. Factoring multivariate polynomials over algebraic number fields. In M.P. Chytil and V. Koubek, editors, *Mathematical Foundations of Computer Science 1984*, volume 176 of *Lecture Notes in Computer Science*, pages 389–396. Springer Berlin Heidelberg, 1984.

[LLL82] A. K. Lenstra, H. W. Lenstra, and L. Lovász. Factoring polynomials with rational coefficients. *Mathematische Annalen*, 261:515–534, 1982. 10.1007/BF01457454.

[LM89] L. Langemyr and S. McCallum. The computation of polynomial greatest common divisors over an algebraic number field. *J. Symb. Comput.*, 8(5):429 – 448, 1989.

[LN02] R. Lidl and H. Niederreiter. *Introduction to finite fields and their applications*. Cambridge Univ. Press, Cambridge, rev. ed., digital print. edition, 2002.

[Luc86] M. Lucks. A fast implementation of polynomial factorization. In *Proceedings of the fifth ACM symposium on Symbolic and algebraic computation*, SYMSAC '86, pages 228–232, New York, NY, USA, 1986. ACM.

[MP11] M. Monagan and R. Pearce. Sparse polynomial multiplication and division in maple 14. *ACM Commun. Comput. Algebra*, 44(3/4):205–209, January 2011.

[Mus71] D. R. Musser. *Algorithms for polynomial factorization.* PhD thesis, 1971.

[Mus75] D. R. Musser. Multivariate polynomial factorization. *J. ACM*, 22(2):291–308, April 1975.

[Nie93] H. Niederreiter. A new efficient factorization algorithm for polynomials over small finite fields. *Applicable Algebra in Engineering, Communication and Computing*, 4:81–87, 1993.

[Ost99] A. Ostrowski. On the significance of the theory of convex polyhedra for formal algebra. *SIGSAM Bull.*, 33(1):5–, March 1999.

[Rob04] X.-F. Roblot. Polynomial factorization algorithms over number fields. *J. Symb. Comput.*, 38(5):1429 – 1443, 2004.

[Sal08] F. K. Abu Salem. An efficient sparse adaptation of the polytope method over $\mathbb{F}_p$ and a record-high binary bivariate factorisation. *J. Symb. Comput.*, 43(5):311–341, 2008.

[Sho] V. Shoup. NTL: A library for doing number theory. `www.shoup.net/ntl`. Version 5.5.2.

[Sho95] V. Shoup. A new polynomial factorization algorithm and its implementation. *J. Symb. Comput.*, 20(4):363–397, 1995.

[SK99] T. Sasaki and F. Kako. Solving multivariate algebraic equation by hensel construction. *Japan Journal of Industrial and Applied Mathematics*, 16:257–285, 1999.

[SS93] T. Sasaki and M. Sasaki. A unified method for multivariate polynomial factorizations. *Japan Journal of Industrial and Applied Mathematics*, 10:21–39, 1993.

[The12] The PARI Group, Bordeaux. *PARI/GP, version* `2.5.2`, 2012. available from `http://pari.math.u-bordeaux.fr/`.

[Tra76] B. M. Trager. Algebraic factoring and rational function integration. In *Proceedings of the third ACM symposium on Symbolic and algebraic computation*, SYMSAC '76, pages 219–226, New York, NY, USA, 1976. ACM.

[Uma08] C. Umans. Fast polynomial factorization and modular composition in small characteristic. In *STOC*, pages 481–490, 2008.

[vH02] M. van Hoeij. Factoring polynomials and the knapsack problem. *Journal of Number Theory*, 95(2):167 – 189, 2002.

[vzG85] J. von zur Gathen. Irreducibility of multivariate polynomials. *J. Comput. Syst. Sci.*, 31(2):225–264, September 1985.

[vzGG99] J. von zur Gathen and J. Gerhard. *Modern computer algebra (1. ed.)*. Cambridge University Press, 1999.

[vzGG02] J. von zur Gathen and J. Gerhard. Polynomial factorization over $\mathbb{F}_2$. *Math. Comput.*, 71(240):1677–1698, 2002.

[vzGG03] J. von zur Gathen and J. Gerhard. *Modern computer algebra (2. ed.)*. Cambridge University Press, 2003.

[vzGK85a] J. von zur Gathen and E. Kaltofen. Factoring sparse multivariate polynomials. *Journal of Computer and System Sciences*, 31(2):265 – 287, 1985.

[vzGK85b] J. von zur Gathen and E. Kaltofen. Factorization of multivariate polynomials over finite fields. *Math. Comp.*, 45(171):251–261, 1985.

[vzGS92] J. von zur Gathen and V. Shoup. Computing frobenius maps and factoring polynomials. *Computational Complexity*, 2:187–224, 1992.

[Wan76] P. S. Wang. Factoring multivariate polynomials over algebraic number fields. *Math. Comp.*, 30(134):pp. 324–336, 1976.

[Wan78] P. S. Wang. An improved multivariate polynomial factoring algorithm. *Math. Comp.*, 32(144):1215–1231, 1978.

[WCF11] W. Wu, J. Chen, and Y. Feng. Sparse bivariate polynomial factorization. `https://sites.google.com/site/jingweichen84/publications`, 2011.

[WR75] P. S. Wang and L. P. Rothschild. Factoring multivariate polynomials over the integers. *Math. Comp.*, 29(131):pp. 935–950, 1975.

[WR76] P. J. Weinberger and L. P. Rothschild. Factoring polynomials over algebraic number fields. *ACM Trans. Math. Softw.*, 2(4):335–350, December 1976.

[Yan09] S. Yang. Computing the greatest common divisor of multivariate polynomials over finite fields. Master's thesis, Simon Fraser University, 2009.

[Yun76] D. Y.Y. Yun. On square-free decomposition algorithms. In *Proceedings of the third ACM symposium on Symbolic and algebraic computation*, SYMSAC '76, pages 26–35, New York, NY, USA, 1976. ACM.

[Zas69] H. Zassenhaus. On hensel factorization, i. *Journal of Number Theory*, 1(3):291 – 311, 1969.

[Zip79] R. E. Zippel. Probabilistic algorithms for sparse polynomials. In *Proceedings of the International Symposiumon on Symbolic and Algebraic Computation*, EUROSAM '79, pages 216–226, London, UK, UK, 1979. Springer-Verlag.

[Zip93] R. E. Zippel. *Effective Polynomial Computation*. Kluwer International Series in Engineering and Computer Science. Kluwer Academic Publishers, 1993.

Wissenschaftlicher Werdegang

2003	Abitur am Kopernikus-Gymnasium, Wissen
seit Oktober 2003	Fernstudium der Mathematik an der TU Kaiserslautern
seit April 2004	Studium der Mathematik mit Nebenfach Wirtschaftswissenschaften an der TU Kaiserslautern
April 2009	Diplom in Mathematik, TU Kaiserslautern
seit September 2009	Doktorand bei Prof. Dr. Gerhard Pfister, TU Kaiserslautern

Curriculum Vitae

2003	Abitur at the Kopernikus-Gymnasium, Wissen
since October 2003	Distance learning of mathematics at the University of Kaiserslautern
since April 2004	Studies of mathematics with minor economics at the University of Kaiserslautern, Germany
April 2009	Diplom in mathematics, University of Kaiserslautern
since September 2009	Ph.D. studies with Prof. Dr. Gerhard Pfister, University of Kaiserslautern