Computing polynomial greatest common divisors using sparse interpolation

by

Hu Jiaxiong

M.Sc., University of Saskatchewan, 2009
B.Sc., University of Saskatchewan, 2007

Thesis Submitted in Partial Fulfillment of the
Requirements for the Degree of
Doctor of Philosophy

in the
Department of Mathematics
Faculty of Science

SIMON FRASER UNIVERSITY
Summer 2018

Approval

Name: **Hu Jiaxiong**

Degree: **Doctor of Philosophy (Mathematics)**

Title: **Computing polynomial greatest common divisors using sparse interpolation**

Examining Committee: **Chair:** Tom Archibald
Professor

Michael Monagan
Senior Supervisor
Professor _____

Petr Lisonek
Supervisor
Professor _____

Nils Bruin
Internal Examiner
Professor _____

George Labahn
External Examiner
Professor _____
David R. Cheriton School of Computer Science
University of Waterloo

Date Defended: **15 August 2018**

Abstract

Computing polynomial greatest common divisors (GCD) plays an important role in Computer Algebra systems because the GCD operation is the bottleneck of many basic applications. For example, to simplify a rational function one divides the numerator and denominator by their GCD. In 1988 Ben-Or and Tiwari introduced the first deterministic polynomial interpolation algorithm which accounts for sparsity. The number of evaluation points needed by the Ben-Or/Tiwari algorithm is linear in the number of non-zero terms in the target polynomial, and moreover, all variables can be interpolated simultaneously hence parallelizing the algorithm is easier. In this thesis, we present modular multivariate polynomial GCD algorithms based on Ben-Or/Tiwari sparse interpolation. They compute the GCD modulo one or more primes. We apply a Kronecker substitution to reduce the number of variables and we modify the Ben-Or/Tiwari evaluation point sequence so that we can use primes of acceptable size (machine primes) as well as gain randomness on the choice of evaluation points to avoid several known issues in polynomial GCD algorithms. Based on several assumptions, we first present a simplified algorithm for GCD computation in $\mathbb{Z}[x_1, \ldots, x_n]$ from which we derive some theoretical bounds and convince the reader why it works. Then we present a practical version of the algorithm where those assumptions are dropped. This leads to a more complicated algorithm but it can be shown that it always terminates and it computes the GCD efficiently. In the 1980s, subsequent research in polynomial GCD algorithm mainly focused on polynomials over number fields. In this thesis, we also present a GCD algorithm for multivariate polynomials in $\mathbb{Q}(\alpha)[x_1, \ldots, x_n]$ where α is an algebraic number. With a prime modulus p, all operations are performed in the finite ring $\mathbb{Z}_p(\alpha)$ where inversions may fail due to zero divisors. We manage to get all necessary bounds to support the correctness of our algorithm.

Keywords: Sparse multivariate polynomial; Polynomial greatest common divisor; Ben-Or and Tiwari interpolation algorithm; Discrete logarithm; Sparse interpolation.

Dedication

I dedicate this thesis to my wife Yun who suffered unspeakable pain during the attainment of this degree.

Acknowledgements

I would like to express my sincere gratitude to my wonderful supervisor Michael Monagan for his patient guidance. Without his constant feedback, this PhD would not have been achievable. I also appreciate his kindness, patience and continuous encouragement because, for personal reasons, the time that my journey takes to complete this degree is long.

I would like to thank the members of my thesis committee: Nils Bruin, George Labahn, Petr Lisonek, and Michael Monagan for reading my thesis, being present at my thesis defense and providing me with useful feedback.

I would like to thank my parents for their unconditional support. I must express my gratitude to Yun, my wife, for her encouragement and continued support throughout the time of my PhD study.

I would like to thank Roman Pearce, Adriano Aarce and Michael Monagan for their coding help.

Finally, I would like to thank the Department of Mathematics at SFU, not only for providing me the funding, but also for giving me the opportunity to attend conferences and meet new people.

Table of Contents

List of Tables

Chapter 1

Introduction

Let A and B be multivariate polynomials in $\mathbb{Z}[x_0, x_1, \ldots, x_n]$. In this thesis we present modular algorithms to compute $G = \gcd(A, B)$, the greatest common divisor (GCD) of A and B over $\mathbb{Z}$. With suitable modification our GCD algorithm is capable of computing GCD over number fields as well. A simple extension case will be discussed.

Let f be a polynomial in variables $x_0, x_1, \ldots, x_n$ and let t be the number of distinct non-zero terms in f and $\deg_{x_i} f$ denote the degree of f in x_i. Let m be the number of possible terms of f in variables $x_0, x_1, \ldots, x_n$ with $\deg_{x_i} \leq d$ for $0 \leq i \leq n$. The polynomial f is called *sparse* if t is much less than m (denoted by $t \ll m$), e.g, $t = \sqrt{m}$. Our algorithm is designed for sparse G and focus on parallelism.

Example 1.1. *Let* $f = x^4y^3zu^2+32x^6u^2+1 \in \mathbb{Z}[x, y, z, u]$. *For polynomials with* 4 *variables and total degree at most* 10*, there are* $\binom{10+4}{10} = 1001$ *possible monomials. But* f *is a three term polynomial hence it is sparse with the degree constraint.*

Since the early age of symbolic computation, efficiently computing a polynomial GCD has been an active research area. Every modern computer algebra system, such as Maple, Mathematica and Magma, must have the capability to compute the GCD because it is the key part in many applications. For example, when computing with fractions of polynomials, GCD computation is the most important operation, which is usually much more expensive than multiplication and division. The efficiency of GCD computation affects the performance of those applications directly. Nowadays modular arithmetic, interpolation algorithms, p-adic lifting, etc are favorite tools in developing multivariate polynomial GCD algorithms.

The most fundamental algorithm to compute the univariate polynomial GCD over $\mathbb{Q}$ is the Euclidean Algorithm which dates back more than two thousand years. However the exponential growth of the largest magnitude of coefficients in intermediate remainders is its major drawback. The Euclidean algorithm generates a sequence of polynomials, each of which is the remainder of the previous two. This is also called a *Euclidean polynomial remainder sequence* (PRS) [Geddes et al., 1992, Section 7.2]. To get a fraction free PRS, we mention the pseudo-division PRS [Geddes et al., 1992, Section 7.2]. But it does not remove

contents at all hence produces the worst coefficient growth. The primitive PRS on the other hand produces the least coefficient growth but the trade-off is to perform many integer GCD computations to remove contents. There are some better variants such as the reduced PRS [Collins, 1967] algorithm and the subresultant PRS algorithm [Brown and Traub, 1971] [Geddes et al., 1992, Section 7.3]. The subresultant PRS is regarded as the best PRS-based algorithm. Its coefficient growth is linear, no GCD computations are required and it can be extended to compute multivariate polynomial GCD over a UFD. However the coefficients in intermediate steps can still grow exponentially in n when being applied to multivariate polynomials. For subresultant PRS, see Section 1.2 for more details. In general those PRS algorithms are regarded as insufficient to compute the GCD of multivariate polynomials.

The most effective tool to solve the coefficient growth problem is modular arithmetic. To be precise, we use a ring homomorphism to map polynomial coefficients from $\mathbb{Z}$ to a simpler domain $\mathbb{Z}_p$ where p is a prime hence all coefficients are contained in the finite field $\mathbb{Z}_p$. Since some information is lost due to the mapping we could reconstruct coefficients in $\mathbb{Z}$ by computing several GCD images over several different finite fields and then lift the coefficients large enough by the Chinese remainder theorem. This method can be generalized to multivariate GCD case by recursively computing the GCD with one less variable by evaluation and then recovering the target GCD by interpolation. Such an evaluation-interpolation style modular GCD algorithm was first introduced by Brown [Brown, 1971]. See also [Geddes et al., 1992, Section 7.4].

Brown's algorithm works effectively for *dense* multivariate polynomial GCD computation. His algorithm handles multivariate polynomials recursively by keeping x_0 as the main variable and evaluate x_n in $\mathbb{Z}_p[x_n][x_0, x_1, \ldots, x_{n-1}]$. So if d is an upper bound for the degree of each variable, $O(d^n)$ points are needed to interpolate the desired GCD G. Brown's algorithm not only provides a way to compute polynomial GCD, but also introduces several techniques which have been widely used subsequently in other applications. Subsequent improvements to Brown's algorithm mainly focused on optimizing the complexity for sparse multivariate polynomials.

A different approach to compute the polynomial GCD was introduced by Char, Geddes and Gonnet. The heuristic algorithm GCDHEU [Char et al., 1989], see also [Geddes et al., 1992, Section 7.7], evaluates input polynomials over $\mathbb{Z}$ rather than $\mathbb{Z}_p$. For the univariate case, it evaluates variables of input polynomials at some carefully chosen integers and then performs a single integer GCD computation. The true polynomial GCD is read from its integer GCD image. The GCDHEU algorithm can be generalized to the multivariate case but it is usually not effective for sparse polynomials.

Another efficient GCD algorithm which is better suited for the multivariate case is the Extended Zassenhaus algorithm (EZ-GCD) [Moses and Yun, 1973]. See also [Geddes et al., 1992, Section 7.6]. It was developed by Moses and Yun in 1973 and named after Zassenhaus to honour his work in number theory. The EZ-GCD algorithm reduces multivariate

polynomials to univariate polynomials by evaluations and modular homomorphisms. Once the GCD is determined in the univariate case, Hensel's lemma is used repeatedly to lift the coefficients and the variables back to multivariate polynomial by generalized Newton iteration. However, the EZ-GCD algorithm comes with several flaws. For example, the bad zero problem which often leads to a large intermediate expression growth of terms, the leading coefficient problem and the non-unit common factor problem. Numerous improvements to the EZ-GCD algorithm have been made. Among those variants, Wang's [Wang, 1980] EEZ-GCD algorithm makes a major improvement. It lifts one variable at a time and the leading coefficient problem is overcome by Wang's heuristic leading coefficient pre-determine algorithm. Wang's EEZ-GCD algorithm is the default polynomial GCD algorithm in many computer algebra systems, such as Axiom, Macsyma, older version of Magma and Maple (prior to version 11). Another notable variant called the sparse EZ-GCD algorithm was developed by Kaltofen in 1985 [Kaltofen, 1985], which makes use of sparse Hensel lifting. We review the EZ-GCD algorithm in Section 1.4.

A milestone probabilistic algorithm for sparse polynomial interpolation was presented by Zippel in 1979 in his PhD thesis [Zippel, 1979a,b]. A direct application is to compute the sparse polynomial GCD of two multivariate polynomials over $\mathbb{Z}$. Zippel implemented his GCD algorithm in Macsyma. Zippel's algorithm is based on the observation that evaluating a non-zero polynomial at a random point will almost never yield zero. It constructs alternating sequences of dense and sparse interpolations by probabilistic techniques. A dense interpolation assumes that all coefficients in a polynomial of degree d are unknowns, hence $d+1$ evaluation points are expected. On the other hand, a sparse interpolation assumes t unknowns where $t \ll d$, so only t evaluation points are needed. The unknowns are determined by solving a $t \times t$ linear system. Zippel's sparse GCD algorithm is similar to Brown's algorithm in structure. They both reduce multivariate polynomials to univariate polynomials. But Zippel's uses $O(ndt)$ distinct evaluation points instead of $O(d^n)$ by Brown's. A deterministic version [Zippel, 1990] of Zippel's algorithm was published in 1990. For the non-monic leading coefficient GCD problem, Zippel's algorithm could fail due to scaling leading coefficient inconsistently. Wang's heuristic leading coefficient algorithm is one available solution but multivariate factorization is required. Another efficient method LINZIP [de Kleine et al., 2005, Wittkopf, 2004] was given by Kleine, Monagan and Wittkopf. It does not require any polynomial factorization and is currently the default multivariate polynomial GCD algorithm used in Maple.

The first deterministic interpolation algorithm which accounts for sparsity was introduced by Ben-Or and Tiwari in 1988 [Ben-Or and Tiwari, 1988]. The algorithm is based on decoding BCH codes. It evaluates $f(2^i, 3^i, 5^i, \ldots, p_n^i)$ for $i = 0, 1, 2, \ldots, 2t-1$. The prime substitutions were motivated by Grigoriev and Karpinsk [Grigoriev and Karpinsk, 1987]. The Ben-Or/Tiwari interpolation is very similar to Prony's method [de Prony, 1795, Giesbrecht et al., 2009] which was developed by Gaspard Riche de Prony in 1795 and used to interpo-

late a univariate function as a sum of exponential functions. The algorithm first recovers all monomials in f then determines all coefficients of f in $\mathbb{Z}$. The Ben-Or/Tiwari algorithm uses $2t$ evaluation points where t is the number of nonzero terms in the target polynomial. However in most cases t is not known and a method to determine t may need more than $2t$ evaluation points. The modular version of the Ben-Or/Tiwari algorithm was first given by Kaltofen, Lakshman and Wiley [Kaltofen et al., 1990]. Although modular arithmetic solves the intermediate expression swell, using a small prime with lifting strategy may fail to determine the correct feedback connect polynomials, which is the key step to recover all monomials. See [Kaltofen et al., 1990, Lemma 1]. In 2000, Kaltofen, Lee and Loo [E Kaltofen and Lobo, 2000] introduced a hybrid version of Zippel's algorithm. Instead of interpolating a variable densely, this algorithm uses the univariate Ben-Or/Tiwari algorithm to race with Newton's interpolation and stops as soon as one of these two algorithms determines the result. The hybrid algorithm uses $O(nt)$ points. However $x_1, x_2, \ldots, x_n$ are still interpolated sequentially due to the structure of Zippel's algorithm. In 2006 Giesbrecht, Labahn and Lee [Giesbrecht et al., 2009] presented a variation of the Ben-Or/Tiwari algorithm for numerical coefficients. Instead of evaluating $x_1, \ldots, x_n$ at prime numbers $2^i, 3^i, \ldots, p_n^i$ they used $\omega_1^i, \omega_2^i, \ldots, \omega_n^i$ as evaluation points where ω_k are roots of unity in $\mathbb{C}$. The monomials can be recovered by computing logarithms numerically.. In 2010 Monagan and Javadi [Javadi and Monagan, 2010] modified the Ben-Or/Tiwari algorithm to interpolate variables in monomials in parallel. Their method requires $O(nt)$ points. In 2010 Kaltofen [Kaltofen, 2010] suggested reducing multivariate interpolation to univariate interpolation by the Kronecker substitution $f(x_1, \ldots, x_n) \to f(x, x^{d+1}, x^{(d+1)^2}, \ldots, x^{(d+1)^{n-1}})$ where $d \geq \deg_{x_i} f$ for $1 \leq i \leq n$. Then the discrete logarithm can be made polynomial in n and $\log d$. He also suggested using a prime $p = 2^k s + 1 > (d+1)^n$ with s small hence an FFT in $\mathbb{Z}_p$ can be used when needed. As far as we know, no one has tried to use the Ben-Or/Tiwari interpolation to compute the polynomial GCD before.

The polynomial GCD algorithm developed by Kaltofen and Trager [Kaltofen and Trager, 1988] should also be mentioned. They introduced the black box representation for symbolic objects. One direct application is the black box GCD algorithm. The black box representation is efficient in space.

All sparse polynomial GCD algorithms mentioned above are difficult to parallelize. The main tool Hensel lifting in EZ-GCD or EEZ-GCD sequentially recovers the GCD one variable at a time, and for each variable, one degree at a time. Zippel's algorithm recovers one variable at a time from univariate GCD images, and each new variable is interpolated densely. So significant speed-up by parallelization is not expected. Still, some parallel implementation work has been done. See Rayes, Wang and Weber [Rayes et al., 1994]. A parallel implementation of the Ben-Or/Tiwari interpolation algorithm is given by Murao and Fujise [Murao and Fujise, 1996].

Subsequent research in polynomial GCD computation mainly focused on number fields. Computing the greatest common divisor over number fields by the Euclidean algorithm tends to be slow since manipulating algebraic numbers directly causes a blowup like in the integer case. In 1974, Rubald [Rubald, 1974] discussed computing GCDs in $\mathbb{Q}(\alpha)[x]$ in his PhD thesis. He also mentioned how to generalize the Brown and Collins's modular GCD algorithm to number fields under certain assumptions. In 1987, Langemyr and McCallum[Langemyr and McCallum, 1989] successfully adapted Brown and Collins's modular algorithm to algebraic number fields. However only a simple extension $\mathbb{Q}(\alpha)$, where α is an algebraic integer, is considered. A probabilistic version was introduced by Langemyr[Langemyr, 1990]. Encarnacion[Encarnacion, 1995] improved Langemyr's algorithm by relaxing the restriction on α from algebraic integers to algebraic numbers. He also used rational number reconstruction to recover the rational coefficients of the GCD instead of scaling GCD images by a denominator bound. In 2002, van Hoeij and Monagan[van Hoeij and Monagan, 2002] presented an algorithm to compute the polynomial GCD over number fields with multiple extensions $\alpha_1, \ldots, \alpha_k$ over $\mathbb{Q}$. Their work was integrated into Maple.

The algorithms above all have a significant impact on the problems of computing the GCD of multivariate polynomials. We want to build a sparse polynomial GCD algorithm which can be highly parallelized. The Ben-Or/Tiwari algorithm interpolates all variables simultaneously which interests us most. In this thesis we present modular polynomial GCD algorithms using Ben-Or/Tiwari interpolation, the discrete logarithm method, the Kronecker substitution and so on. Partial results have been published in [Hu and Monagan, 2016]. The timing results are very promising. Let $\#f$ denote the number of distinct terms in polynomial f. For our benchmark problem where $\#G \approx 10^4$, $\#A \approx 10^6$ and $\#B \approx 10^6$, Maple takes 22,111 seconds, Magma takes 1,611 seconds, and our new algorithm takes 4.67 seconds on 16 cores and 48.17 seconds on 1 core. See Table 7.2 for more timing results.

Before we present our new GCD algorithms it is necessary to reveal more details about several algorithms mentioned above by examples so that readers would be more familiar with various techniques introduced by those algorithms so that they may better understand the evolution of multivariate polynomial GCD algorithms.

We first fix some notations which are used frequently. Let $A, B \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$ and $G = \gcd(A, B)$. $\bar{A} = A/G$ and $\bar{B} = B/G$ are the cofactors of A and B respectively. For $f \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$, $LC(f)$ denotes the leading coefficient of f taken in x_0, $\#f$ denotes the number of non-zero terms in f and $Supp(f)$ denotes the support of f, which is the set of all monomials appearing in f.

1.1 The Euclidean algorithm

The Euclidean algorithm is the foundation of GCD computation. All GCD algorithms mentioned above execute the Euclidean algorithm at some point. But the rapid growth of the size

of coefficients in the Euclidean algorithm is a well-known problem. The following example demonstrates this issue.

Example 1.2. *Consider the GCD of the following two univariate polynomials:*

$$A(x) = 6x^4 + 9x^3 + 5x^2 + x + 10 \quad \textit{and} \quad B(x) = 3x^3 + 5x^2 + 4x + 10.$$

Let $r_1 = A$ and $r_2 = B$. The Euclidean algorithm executes as follows.

Dividend	Divisor	Quotient	Remainder
r_1	r_2	$q_3 = 2x - \frac{1}{3}$	$r_3 = -\frac{4}{3}x^2 - \frac{53}{3}x + \frac{40}{3}$
r_2	r_3	$q_4 = -\frac{9}{4}x + \frac{41}{16}$	$r_4 = \frac{7911}{16}x - \frac{675}{2}$
r_3	r_4	$q_5 = -\frac{64}{23733}x - \frac{87088}{2317923}$	$r_5 = \frac{168160}{257547}$
r_4	r_5	$q_6 = \frac{2037454317}{2690560}x - \frac{34768845}{67264}$	$r_6 = 0$

Table 1.1: Euclidean algorithm

The example shows two problems: the magnitude of coefficients in the remainders grows exponentially and the coefficients are fractions instead of integers. If each remainder is made to be monic then the growth of the magnitude of coefficients will be linear in degree of inputs but it requires many integer GCD computation at each step. In order to better understand those problems, let us first review polynomial pseudo division [Geddes et al., 1992, Section 2.7]. For the algorithm, see Figure 1.1. Let $k = \deg_x A - \deg_x B + 1$ and $LC(B)$ be the leading coefficient of B. Instead of computing the quotient q and remainder r so that $A = Bq + r$, we compute $\breve{q}, \breve{r} \in \mathbb{Z}[x]$ so that $LC(B)^k A = B\breve{q} + \breve{r}$ and $\breve{q} = pquo(A, B)$ and $\breve{r} = prem(A, B)$ are called *pseudo quotient* and *pseudo remainder* respectively. We first give the formal definition of a polynomial remainder sequence.

Definition 1.1. Let R be a UFD and $A(x)$ and $B(x)$ be polynomials in $R[x]$. Suppose $\deg_x A \geq \deg_x B$. A polynomial remainder sequence (PRS) for A and B is a sequence of polynomials $r_1(x), r_2(x), \ldots, r_k(x)$ in $R[x]$ satisfying

1. $r_1(x) = A(x)$, $r_2(x) = B(x)$,
2. $a_i r_{i-1}(x) = q_i(x) r_i(x) + b_i r_{i+1}(x)$ with $a_i, b_i \in R$ and $q_i(x) \in R[x]$,
3. $prem(r_{k-1}, r_k) = 0$.

Pseudo-Division
Input: $A = \sum_{i=0}^{m} a_i x^i$ and $B = \sum_{j=0}^{n} b_j x^j$ where $a_i, b_i \in R$ where R is a commutative ring with 1. We assume $m \geq n$ and $b_n \neq 0$.
Output: $q, r \in R[x]$ such that $b_n^{m-n+1} A = qB + r$.

1 Set $r = A$, $q = 0$ and $k = 0$.
2 While $r \neq 0$ and $\deg_x r \geq n$ do
  3 Set $dr = \deg_x r$ and $lr = LC(r)$.
  4 Set $r = b_n r - lr \cdot x^{dr-n} B$.
  5 Set $q = b_n q + lr \cdot x^{dr-n}$.
  6 Set $k = k + 1$.
7 RETURN $q = b_n^{m-n+1-k} q$ and $r = b_n^{m-n+1-k} r$.

Figure 1.1: Pseudo Division Algorithm

Let us focus on $R = \mathbb{Z}$ for now. In Definition 1.1 the larger the b_i the smaller the magnitude of the coefficients in r_{i+1} is and vice versa. If we use the pseudo-division to replace the normal division in the Euclidean algorithm, we get the pseudo PRS algorithm where $a_i = r_i^{\deg_x r_{i-1} - \deg_x r_i + 1}$ and $b_i = 1$. No fractions appear in the pseudo PRS, but since $b_i = 1$ the growth of the coefficients of remainders could be fast. Let us see the following example.

Example 1.3. *For polynomials $A(x), B(x) \in \mathbb{Z}[x]$ considered in Example 1.2, we use pseudo remainder PRS to determine their polynomial GCD. Let $r_1 = LC(B)^{4-3+1} A$ and $r_2 = B$, the PRS is showed in the following table.*

Remainder	Quotient
$r_1 = LC(B)^{4-3+1} A$	
$r_2 = B$	
$\check{r_3} = -12x^2 - 159x + 120$	$\check{q_3} = 18x - 3$
$\check{r_4} = 71199x - 48600$	$\check{q_4} = -36x + 417$
$\check{r_5} = 29789039520$	$\check{q_5} = -854388x - 11903841$
$\check{r_6} = 0$	$\check{q_6} = 2120949824784480x - 1447747320672000$

Table 1.2: Pseudo PRS

The severe coefficient growth issue shown in above example can be solved by the *primitive PRS*.

Definition 1.2. Let $f(x) = \sum_{i=0}^{d} a_i x^i \in \mathbb{Z}[x]$ be a non-zero polynomial. The integer content of f, denoted by $icont(f)$, is defined $icont(f) = \gcd(a_0, a_1, \cdots, a_d)$. The primitive part of f is denoted as $pp(f) = f/icont(f)$.

In each pseudo division step, we replace the remainder by its primitive part. The following example demonstrates this method.

Example 1.4. *For polynomials $A(x), B(x) \in \mathbb{Z}[x]$ considered in Example 1.2 and Example 1.3, we use primitive PRS to determine their polynomial GCD. Let $r_1 = LC(B)^{4-3+1} pp(A)$ and $r_2 = pp(B)$, then we have*

Dividend	Divisor	Pseudo remainder	Primitive part
r_1	r_2	$\check{r_3} = -12x^2 - 159x + 120$	$pp(\check{r_3}) = -4x^2 - 53x + 40$
$LC(pp(\check{r_3}))^{3-2+1} r_2$	$pp(\check{r_3})$	$\check{r_4} = 7911x - 5400$	$pp(\check{r_4}) = 293x - 200$
$LC(pp(\check{r_4}))^{2-1+1} pp(\check{r_3})$	$pp(\check{r_4})$	$\check{r_5} = 168160$	$pp(\check{r_5}) = 1$
$LC(pp(\check{r_5}))^{2-1+1} pp(\check{r_4})$	$pp(\check{r_3})$	$\check{r_6} = 0$	$pp(\check{r_6}) = 0$

Table 1.3: Primitive PRS

The magnitude of coefficients of remainders in the primitive PRS is well controlled and in fact it is the best possible. The trade-off is to perform many integer GCD computations to remove the contents of remainders.

When we compare the original Euclidean algorithm with the pseudo PRS, it seems nice to compute GCD in $R[x]$ using only arithmetic from the domain $R[x]$, rather than working with the quotient field of R as we saw in Example 1.2. That is one advantage of PRS algorithms using pseudo division. Moreover, PRS algorithms compute a univariate polynomial over a UFD hence it could be generalized to the multivariate case to compute GCD in $\mathbb{Z}[x_0, x_1, \ldots, x_n]$ as we treat polynomials to be in $\mathbb{Z}[x_1, \ldots, x_n][x_0]$, see Example 1.7. However the computational cost to remove the polynomial content could be high because it involves many recursive multivariate GCD computations which causes a growth in intermediate degrees. Therefore a PRS algorithm with linear growth of coefficients and no GCD calculation is preferred. The best option so far is the subresultant PRS which will be discussed in the next section.

1.2 Subresultant PRS

We are interested in a PRS which keeps the coefficient growth under control and avoids GCD computations to remove contents. The *reduced PRS* is such a method that was discovered by Sylvester [Sylvester, 1853] and rediscovered by Collins [Collins, 1967]. It picks $a_i = LC(r_i)^{\deg_x r_{i-1} - \deg_x r_i + 1}$ and $b_i = a_{i-1}$ where $3 \leq i \leq k$. The reduced PRS is suitable for the *complete* PRS which requires $\deg_x r_{i-1} - \deg_x r_i = 1$. Otherwise the coefficient growth can still be exponential in degrees of input polynomials. For an incomplete PRS case, the subresultant PRS is recommended. Sylvester initiated subresultant PRS research in [Sylvester, 1853]. Habicht in [Habicht, 1948] made continued efforts on this topic. In the late sixties and early seventies, Brown [Brown and Traub, 1971] and Collins [Collins, 1967] independently simplified the formation of the subresultant PRS. Brown observes that the subresultant PRS requires exponential time to compute the GCD of sparse polynomials. But it is still the best Euclidean based PRS algorithm we can get. We consider $A, B \in \mathbb{Z}[x]$

for now but the subresultant PRS works for $A, B \in D[x]$ where D is an integral domain which will be discussed in Section 8.5.

Definition 1.3. Suppose $\deg_x A \geq \deg_x B$. Let $r_1 = A, r_2 = B$ and $\delta_i = \deg_x r_{i-1} - \deg_x r_i$. A subresultant PRS $[r_1, r_2, \ldots, r_k]$ satisfies $a_i r_{i-1}(x) = q_i(x) r_i(x) + b_i r_{i+1}(x)$ where

$$a_i = LC(r_i)^{\delta_i+1},\ b_2 = -1^{\delta_2+1},\quad \psi_2 = -1,$$

$$\psi_i = -LC(r_{i-1})^{\delta_{i-1}} / \psi_{i-1}^{\delta_{i-1}-1},\quad b_i = -LC(r_{i-1})\psi_i^{\delta_i} \ \text{ for } \ 3 \leq i \leq k.$$

The proof that the subresultant PRS is a valid PRS is difficult. See [Geddes et al., 1992, Section 7.3]. First let us check the following example.

Example 1.5. *Consider polynomials $A(x), B(x) \in \mathbb{Z}[x]$ in Example 1.2 where*

$$A(x) = 6x^4 + 9x^3 + 5x^2 + x + 10 \quad \textit{and} \quad B(x) = 3x^3 + 5x^2 + 4x + 10.$$

Let $r_1 = A$, $r_2 = B$. The subresultant PRS is

i	δ_i	ψ_i	remainder
1			$r_1 = A$
2	1	-1	$r_2 = B$
3	1	-3	$r_3 = -12x^2 - 159x + 120$
4	1	12	$r_4 = 7911x - 5400$
5	1	-7911	$r_5 = 2553930$
6			$r_6 = 0$

Table 1.4: Subresultant PRS

The magnitude of coefficients in remainders is considerably less than we have in the pseudo-division PRS. In fact Brown [Brown and Traub, 1971] observed that the growth of the coefficients was linear in the number of division steps. More importantly there is no GCD computation. The subresultant PRS could also be computed by the definition of the subresultant polynomial.

Let $A = \sum_{i=1}^{n} a_i x^i \in \mathbb{Z}[x]$ and $B = \sum_{i=1}^{m} b_i x^i \in \mathbb{Z}[x]$. The Sylvester matrix of A and B is

$$M = \begin{bmatrix} a_n & 0 & & 0 & b_m & 0 & & 0 \\ a_{n-1} & a_n & & 0 & b_{m-1} & b_m & & 0 \\ \vdots & a_{n-1} & \ddots & 0 & \vdots & b_{m-1} & \ddots & 0 \\ a_1 & \vdots & & a_n & b_1 & \vdots & & b_m \\ a_0 & a_1 & & a_{n-1} & b_0 & b_1 & & b_{m-1} \\ 0 & a_0 & & \vdots & 0 & b_0 & & \vdots \\ 0 & 0 & \ddots & a_1 & 0 & 0 & \ddots & b_1 \\ 0 & 0 & & a_0 & 0 & 0 & & b_0 \end{bmatrix}. \tag{1.1}$$

The resultant of A and B is $res_x(A,B) = \det M \in \mathbb{Z}$. Let $M(i,j)$ be the $(m+n-2j) \times (m+n-2j)$ submatrix of M obtained by deleting

1. columns $m-j+1$ to m;
2. columns $n+m-j+1$ to $n+m$;
3. rows $n+m-2j$ to $n+m$ except row $n+m-i-j$.

Definition 1.4. The j-th *subresultant polynomial* of A and B is the polynomial of degree j defined as

$$S(j,A,B) = \det M(0,j) + \det M(1,j)x + \cdots + \det M(j,j)x^j.$$

It is clear that $S(0,A,B) = \det M(0,0) = res_x(A,B)$. In fact, for $d \geq 0$, $\deg_x \gcd(A,B) = d$ if and only if $S(k,A,B) = 0$ for $0 \leq k \leq d-1$ and $S(d,A,B) \neq 0$. We use the subresultant polynomial definition to redo Example 1.5.

Example 1.6. *We compute the j-th subresultant polynomial of A and B in Example 1.5 according to Definition 1.4.*

$$A(x) = 6x^4 + 9x^3 + 5x^2 + x + 10 \quad \textit{and} \quad B(x) = 3x^3 + 5x^2 + 4x + 10.$$

The Sylvester matrix of A and B is

$$\begin{bmatrix} 6 & 0 & 0 & 3 & 0 & 0 & 0 \\ 9 & 6 & 0 & 5 & 3 & 0 & 0 \\ 5 & 9 & 6 & 4 & 5 & 3 & 0 \\ 1 & 5 & 9 & 10 & 4 & 5 & 3 \\ 10 & 1 & 5 & 0 & 10 & 4 & 5 \\ 0 & 10 & 1 & 0 & 0 & 10 & 4 \\ 0 & 0 & 10 & 10 & 0 & 0 & 10 \end{bmatrix}.$$

Since $\deg_x B(x) = 3$*, we have*

$$\begin{aligned} S(0,A,B) &= 2553930; \\ S(1,A,B) &= 7911x - 5400; \\ S(2,A,B) &= -12x^2 - 159x + 120; \\ S(3,A,B) &= 3x^3 + 5x^2 + 4x + 10 = B. \end{aligned}$$

We observe that the same result is obtained as we have in Example 1.5 with the order reversed.

Example 1.5 demonstrates a complete PRS case. If the PRS is incomplete, then the the subresultant PRS $[r_3, r_4, \ldots, r_k]$ computed according to Definition 1.3 is a subset of the subresultant polynomials $[S(j, f_1, f_2) : 0 \leq j \leq \deg_x f_2 - 1]$ computed according to Definition 1.4. See [Geddes et al., 1992, Example 7.6, Example 7.9]. For the relation between r_i and $S(j, f_1, f_2)$, we refer to the Fundamental Theorem of PRS [Geddes et al., 1992, Section 7.3].

The subresultant PRS algorithm is regarded as the best PRS algorithm to control the coefficient growth as well as being a fraction-free calculation. It can also be used to compute GCD of multivariate polynomials over $\mathbb{Z}$. Now let us consider the subresultant PRS in $\mathbb{Z}[x_1, \ldots, x_n]$ with respect to some monomial order. Let x_1 be the main variable hence the coefficient domain is $\mathbb{Z}[x_2, \ldots, x_n]$. The definition of the subresultant PRS is identical to Definition 1.3 except we change the coefficient domain. We need to define the content and the primitive part of a multivariate polynomial. If D is a UFD, then the polynomial domain $D[x_1, \ldots, x_n]$ is also a UFD. Normally we want to impose an additional condition on GCDs of polynomials in $D[x_1, \ldots, x_n]$ in order to make it unique. For example, for $a, b \in D[x_1, \ldots, x_n]$, a and b are associates if and only if $a = ub$ for some unit $u \in D[x_1, \ldots, x_n]$. Then a and b are in the same associate class. The element which is chosen to represent an associate class is said to be *unit normal*. In $\mathbb{Z}$, all non-negative integers are defined to be unit normal. In any field, every pair of non-zero elements are associates hence we define 0 and 1 to be unit normal.

Definition 1.5. Let D be a UFD and $D[x_1, \ldots, x_n]$ be a polynomial domain with respect to some monomial order. Assume x_1 is the main variable. Let $f \in \sum_{k=0}^{m} a_k x_1^k \in D[x_1, \ldots, x_n]$. Hence $a_k \in D[x_2, \ldots, x_n]$. The *content* of f in x_1, denoted by $cont(f(x_1))$, is the unique unit normal $\gcd(a_1, a_2, \ldots, a_m)$. The *primitive part* of f is $f/cont(f(x_1))$ and we say f is *primitive* if $\mathrm{cont}(f(x_1)) = 1$.

We note that the input polynomials should be primitive with respect to the main variable because PRS algorithms always return the primitive part of the last non-zero remainder with respect to the main variable as the GCD. If inputs are not primitive then we compute the GCD of the primitive parts of inputs.

Example 1.7. *Consider the GCD problem*

$$A = 18x^7y + 15x^6 + 12x^3y + 10x^2, \quad B = 15x^6y^2 + 27x^4y^4 + 10x^2y^2 + 18y^4.$$

Let $r_1 = A$ and $r_2 = B$. According to Definition 1.3, we have

$$\begin{aligned} r_3 =& -7290x^5y^7 - 6075x^4y^6 - 4860xy^7 - 4050y^6. \\ r_4 =& 6377292x^4y^{14} + 2460375x^4y^{10} + 4251528y^{14} + 1640250y^{10}. \\ r_5 =& 0. \end{aligned}$$

Since A is primitive with respect to x, the primitive part of r_4 is $3x^4+2$ which is the GCD of A and B.

Comparing r_4 and $\gcd(A,B)$, r_4 is more dense. The degree of y and the magnitude of coefficients both significantly increase during the computation.

1.3 The GCDHEU algorithm

We want to mention this algorithm because it computes polynomial GCD from a different perspective which first reduces the problem to GCD computation over $\mathbb{Z}$ and then recovers the polynomial GCD from its integer GCD image. A brief review is given in this section. See [Char et al., 1989] for more details.

Suppose $A(x), B(x) \in \mathbb{Z}[x]$ and $g(x) = \gcd(A(x), B(x)) = \sum_{i=0}^{d} c_i x^i$. Let $\beta \in \mathbb{Z}$ be a positive integer which bounds twice the magnitudes of all coefficients in $A(x), B(x)$ and in any of their factors. Hence β is twice larger than the magnitudes of the coefficients in $g(x)$. We first compute $\alpha = \gcd(A(\beta), B(\beta))$ and then convert α to $g(x)$ by determining its β-adic representation with the *symmetric representation of coefficients*, that is $\alpha = c_0 + c_1\beta + c_2\beta^2 + \cdots + c_k\beta^d$. Then $g(x)$ is obtained by substituting β for x. We illustrate this algorithm with the following example.

Example 1.8. *Suppose we want to find the GCD of the following two univariate polynomials*

$$A = 6x^4 - 14x^3 + 19x^2 - 21x + 15 \quad \textit{and} \quad B = 9x^3 - 15x^2 + x + 10.$$

$\beta = 100$ is assumed to be larger than twice the magnitudes of the coefficients in A, B and $G = \gcd(A,B)$.

$$\gamma = \gcd(f_1(100), f_2(100)) = \gcd(586187915, 8850110) = 29305.$$

Now we compute the symmetric 100-adic representation of 29305*.*

$$\begin{aligned} c_0 &= \gamma \mod 100 = 5, \quad \gamma = \frac{\gamma - c_0}{100} = 293. \\ c_1 &= \gamma \mod 100 = -7, \quad \gamma = \frac{\gamma - c_1}{100} = 3. \\ c_2 &= \gamma \mod 100 = 3, \quad \gamma = \frac{\gamma - c_2}{100} = 0. \end{aligned}$$

Therefore $\gamma = 5 - 7 \cdot 100 + 3 \cdot 100^2$. Once we substitute the base 100 *with x we have the GCD $G = 5 - 7x + 3x^2$.*

One problem is that the magnitude of the coefficients in $g(x)$ is not given hence some estimation is required. Let h_1 and h_2 be the largest magnitudes of the coefficients in A and B respectively. Algorithm GCDHEU picks $\beta > 1 + 2\min(h_1, h_2)$. Suppose G is the result

converted from α, Theorem 2 of [Char et al., 1989] shows that $G = \gcd(A, B)$ if and only if G divides A and B. The GCDHEU algorithm can be generalized to multivariate GCD computation by recursively evaluating each variable at an integer. Let us redo Example 1.7. For the algorithm, see [Geddes et al., 1992, Section 7.7].

Example 1.9. *Consider the GCD problem*

$$A = 54x^7y + 45x^6 + 36x^3y^2 + 30x^3y + 30x^2y + 25x^2,$$

$$B = 45x^6 + 81x^4y^2 + 30x^2y + 54y^3 + 25x^2 + 45y^2.$$

Since the last term of B is $45y^2$, their GCD must be primitive in x. GCDHEU is integer content sensitive because it performs calculation over $\mathbb{Z}$ and therefore integer contents need to be removed in each recursive call. The integer contents of A and B are both 1. Therefore no content needs to be removed. We set $A_1 = A$ and $B_1 = B$. Since the heights of A_1 and B_1 are $\|A_1\| = 54$ and $\|B_1\| = 81$ respectively, we pick $\beta = 2 + 2\min(\|A_1\|, \|B_1\|) = 110$ as the evaluation point for x. Let

$$A_2 = A_1(\beta, y) = 47916000y^2 + 10523072380293000y + 79720245302500 \text{ and}$$

$$B_2 = B_1(\beta, y) = 54y^3 + 11859210045y^2 + 363000y + 79720245302500.$$

The integer contents of A_2 and B_2 are 60500 and 1 respectively. Therefore we remove the integer content of A_2 to get A_3 and set $B_2 = B_3$ and have

$$A_3 = 792y^2 + 173935080666y + 1317690005,$$

$$B_3 = 54y^3 + 11859210045y^2 + 363000y + 79720245302500.$$

Since $\|A_3\| = 173935080666$ and $\|B_3\| = 79720245302500$, the evaluation point for y is $\alpha = 2 + 2\min(\|A_3\|, \|B_3\|) = 347870161334$ and $\gcd(A_3(\alpha), B_3(\alpha)) = 2088538658009$. The α-adic expansion of 2088538658009 is $1317690005y^0 + 6y^1$. We multiply $1317690005y^0 + 6y^1$ by 1 which is the GCD of the integer contents of A_2 and B_2. The β-adic expansion of 6 is $6x^0$ hence the coefficient of y^1 is 6. The β-adic expansion of 1317690005 is $5x^0 + 0x^1 + 0x^2 + 0x^3 + 9x^4$ hence the coefficient of y^0 is $9x^4 + 5$ which divides A and B. Therefore $\gcd(A, B) = 9x^4 + 6y + 5$.

For $A, B \in \mathbb{Z}[x_1, x_2, \ldots, x_n]$, let $d > \max_{i=1}^{n}(\deg_{x_i} A, \deg_{x_i} B)$ and let $x_1 = \beta$. Then we recursively call GCDHEU to compute $\gcd(A(\beta, x_2, \ldots, x_n), B(\beta, x_2, \ldots, x_n))$ with one less variable. The magnitude of coefficients in $A(\beta, x_2, \ldots, x_n)$ and $B(\beta, x_2, \ldots, x_n)$ increases by a factor β^d. Therefore the magnitude of integers at the base recursive case has size $O(\beta^{d^n})$. Obviously the algorithm is not acceptable for sparse GCD problems. When this algorithm was published approximately 30 years ago, the authors suggested that GCDHEU was often

successful for GCD problems up to four variables. Prior to Maple 11, Maple used GCDHEU for small and dense problems. For larger and sparse problems, the Maple implementation used Wang's EEZ-GCD algorithm which we discuss later.

1.4 The EZ-GCD algorithm

The EZ-GCD algorithm maps GCD problems in $\mathbb{Z}[x_0, \ldots, x_n]$ to $\mathbb{Z}_p[x_0]$ by evaluation homomorphism (modulo $I = \langle p, x_1 - \alpha_1, x_2 - \alpha_2, \ldots, x_n - \alpha_n \rangle$) and modular homomorphism (modulo a prime p). It uses a technique called Hensel lifting [Geddes et al., 1992, Section 6.4] to rebuild the result. To be precise, it first performs a p-adic iteration to lift coefficients of the univariate GCD image from $\mathbb{Z}_p$ to $\mathbb{Z}_{p^l}$ for some positive integer l, and then lift the univariate GCD image from $\mathbb{Z}_{p^l}[x_0]$ back to $\mathbb{Z}_{p^l}[x_0, \ldots, x_n]$ by I ideal-adic iteration. Both lifting methods are derived from Hensel's lemma. We first give an example to demonstrate the idea.

Example 1.10. *We want to determine the GCD of the following two polynomials*

$$A = y^2 + 2xy - 3y + x^2 - 3x - 4 \quad \textit{and} \quad B = y^2 + 2xy - 4y + x^2 - 4x - 5.$$

Using $p = 17$ *and* $y = 0$*, we have*

$$A(x, 0) \mod p = x^2 + 14x + 13 \quad \textit{and} \quad B(x, 0) \mod p = x^2 + 13x + 12.$$

Next we compute

$$g = \gcd(A(x, 0), B(x, 0)) = x + 1 \mod p.$$

The cofactor of g *is* $\bar{f}_1(x, 0) = f_1(x, 0)/G = x + 13$*. Hence*

$$A(x, y) \equiv (x + 1)(x + 13) \mod \langle y, p \rangle.$$

We first use the p*-adic iteration to lift the coefficients and obtain*

$$A(x, y) \equiv (x + 1)(x - 4) \mod \langle y, p^2 \rangle.$$

Then we use the ideal-adic iteration to lift the variable y *in both factors and obtain*

$$A(x, y) \equiv (x + 1 + y)(x - 4 + y) \mod \langle y^2, p^2 \rangle.$$

Since $x + y + 1$ *divides* f_1 *and* f_2 *over* $\mathbb{Z}$ *we have*

$$\gcd(A, B) = x + y + 1.$$

We briefly describe the p-adic univariate Hensel lifting here so that readers better understand how to get to the key lifting step which is to solve a polynomial Diophantine equation. For example, for given polynomials a, b, c we want to determine s and t so that $sa + tb = c$ which is called the Diophantine equation. Suppose p is a prime, $f(x) \in \mathbb{Z}[x]$, $f(x) = a(x)b(x)$ and $\gcd(a(x), b(x)) = 1$. Let $a(x) \mod p^l = u(x)$ and $b(x) \mod p^l = v(x)$. We also assume $f(x) = u_0(x)v_0(x) \mod p$ and $\gcd(u_0, v_0) = 1$ where $u(x) \mod p = u_0(x)$, $v(x) \mod p = v_0(x)$. And we want to lift $u_0(x)$ to $u(x) \in \mathbb{Z}_{p^l}[x]$ and $v_0(x)$ to $v(x) \in \mathbb{Z}_{p^l}[x]$. The p-adic representations of $u(x)$ and $v(x)$ are

$$u(x) = u_0 + u_1 p + u_2 p^2 + \cdots + u_l p^{l-1} \quad \text{and} \quad v(x) = v_0 + v_1 p + v_2 p^2 + \cdots + v_l p^{l-1}.$$

We also define $u^{(i)} = u(x) \mod p^i$, $v^{(i)} = v(x) \mod p^i$ for $2 \leq i \leq l$, $u^{(1)} = u_0$ and $v^{(1)} = v_0$. Assume we have obtained the factorization $f(x) = u^{(i)}v^{(i)} \mod p^i$ and want to determine $f(x) = u^{(i+1)}v^{(i+1)} \mod p^{i+1}$ which can be expressed as

$$\begin{aligned} f - u^{(i+1)}v^{(i+1)} =& f - (u^{(i)} + u_{i+1}p^i)(v^{(i)} + v_{i+1}p^i) \\ =& f - u^{(i)}v^{(i)} - (u^{(i)}v_{i+1}p^i + v^{(i)}u_{i+1}p^i) \mod p^{i+1} \end{aligned}$$

We want to determine $u^{(i+1)}$ and $v^{(i+1)}$ so that $f - u^{(i+1)}v^{(i+1)} = 0 \mod p^{i+1}$. From the previous equation we have

$$f - u^{(i)}v^{(i)} - (u^{(i)}v_{i+1}p^i + v^{(i)}u_{i+1}p^i) = 0 \mod p^{i+1}$$

which is equivalent to

$$\frac{f - u^{(i)}v^{(i)}}{p^i} - (u^{(i)}v_{i+1} + v^{(i)}u_{i+1}) = 0 \mod p.$$

Since $u^{(i)} = u_0 \mod p$ and $v^{(i)} = v_0 \mod p$, we have

$$u_0 v_{i+1} + v_0 u_{i+1} = C \mod p, \tag{1.2}$$

where $C = \frac{f - u^{(i)}v^{(i)}}{p^i} \mod p$. In Equation (1.2), u_0, v_0 and C are known, u_{i+1} and v_{i+1} can be determined by solving the Diophantine equation (1.2). Therefore $u^{(i+1)}$ and $v^{(i+1)}$ can be determined. In order to get unique solutions we also impose $\deg_x v_{i+1} < \deg_x v_0$ and $\deg_x u_{i+1} < \deg_x u_0$. Since $\gcd(u_0, v_0) = 1 \mod p$, by Theorem 2.6 of [Geddes et al., 1992], $u^{(i+1)}$ and $v^{(i+1)}$ are unique.

The I ideal-adic iteration is a multivariate Hensel lifting. The moduli I^i contain finite sums of polynomial $l_1 l_2 \cdots l_i$ where $l_k \in \{x_1 - \alpha_1, x_2 - \alpha_2, \ldots, x_n - \alpha_n\}$ for $1 \leq k \leq i$. For

example, if $n = 3$, then

$$I^2 = \langle (x_1-\alpha_1)^2, (x_2-\alpha_2)^2, (x_3-\alpha_3)^2, (x_1-\alpha_1)(x_2-\alpha_2), (x_1-\alpha_1)(x_3-\alpha_3), (x_2-\alpha_2)(x_3-\alpha_3)\rangle.$$

Therefore I-adic representations of factors of a multivariate polynomial are complicated. Besides, we have to solve a multivariate Diophantine equation. Since Hensel lifting is not the tool we use in our new algorithm, we recommend readers to check [Geddes et al., 1992, Wang, 1980, 1978, Monagan and Tuncer, 2016].

Let $A, B \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$, $G = \gcd(A, B)$, $a = A \mod \langle I, p\rangle$ and $b = B \mod \langle I, p\rangle$ and $g = \gcd(a, b)$. Let $\bar{a}$ and $\bar{b}$ be the cofactors of a and b respectively hence $a = \bar{a}c$ and $b = \bar{b}g$. The EZ-GCD proceeds based on the assumption that $\gcd(g, \bar{a}) = 1$ or $\gcd(g, \bar{b}) = 1$. In particular if $\gcd(g, \bar{a}) = 1$ then we construct $A = G\bar{A}$ from $a = g\bar{a}$ by Hensel lifting where $\bar{A}$ is the cofactor of A.

There are several problems in the EZ-GCD algorithm. The algorithm tries to use as many zero evaluation points for α_k as possible. In some circumstances, zero evaluations cause the leading term to vanish in which case a non-zero evaluation point must be used. For instance, if all α_k are zeros then we lift $G(x_0, 0, \ldots, 0)$ back to $G(x_0, x_1, \ldots, x_n)$ which is the case we prefer. At the other extreme, if we cannot use zero as evaluation points, that is $\alpha_k \neq 0$ for $1 \leq k \leq n$, then we have to lift $G(x_0, \alpha_1, \ldots, \alpha_n)$ back to $G(x_0, x_1 - \alpha_1, \ldots, x_n - \alpha_n)$. In the intermediate steps we would have terms like $(x_1 - \alpha_n)^{e_1} \cdots (x_n - \alpha_n)^{e_n}$ instead of $x_1^{e_1} \cdots x_n^{e_n}$. Expanding that term could cause large intermediate expression growth. This is regarded as the bad zero problem.

In the EZ-GCD algorithm the leading coefficient is only correct up to units. Hence it cannot handle a non-monic GCD problem correctly. We can use the normalization technique to scale GCD images by $LC(A)$ but it could produce an expression swell. Wang [Wang, 1980] solves this problem by his heuristic leading coefficient algorithm which the efficiency of the EEZ-GCD algorithm depends mainly on. See Section 1.7 for details.

It is also possible that $\gcd(g, \bar{a}) \neq 1$ and $\gcd(g, \bar{b}) \neq 1$ which is called the common divisor problem. Wang overcomes this issue by diving out the offending common factor. Suppose $C \in \mathbb{Z}[x_0, \ldots, x_n]$ is the common factor so that $\gcd(g, \bar{a}) = C$. Then we have $\bar{a} \equiv \bar{a}_0 C \mod I$ and $g \equiv g_0 C \mod I$ where $\bar{a}_0 \equiv \bar{a}/C \mod I$ and $g_0 \equiv g/C \mod I$. Instead of lifting $A \equiv g\bar{a} \mod I$, Wang uses the parallel version of the Hensel's lemma to lift

$$A \equiv g_0 \bar{a}_0 C^2 \mod I.$$

1.5 Brown's algorithm

Modular arithmetic is the best tool to control the integer coefficient growth because all coefficients of polynomials sit in a finite field. Brown's algorithm [Brown, 1971] is the first efficient modular GCD algorithm designed to handle the multivariate case. Brown's algo-

rithm has made a profound impact on the design of modern polynomial GCD algorithms. Several techniques it introduced have become standard in this subject. Let us consider the following example.

Example 1.11. *Consider the GCD of the following two polynomials in* $\mathbb{Z}[x, y]$.

$$A = (5xy^2 + x^2 - 18)(x + 3y + 11),$$

$$B = (5xy^2 + x^2 - 18)(x + 3y).$$

Note A *and* B *are primitive polynomials in* x *because they are monic in* x. *Let* $p = 11$ *be the first prime. We evaluate* A *and* B *at* $y = 1$ *modulo* 11 *and obtain*

$$A(x, 1) = x^3 + 8x^2 + 8x + 1 \mod 11, \qquad B(x, 1) = x^3 + 8x^2 + 8x + 1 \mod 11.$$

Now we compute

$$\gcd(A(x, 1), B(x, 1)) = x^3 + 8x^2 + 8x + 1 \mod 11$$

where the GCD is computed in $\mathbb{Z}_p[x]$ *using the Euclidean algorithm. We run into a problem because the degree of the GCD in* x *is 3 which is the same as* $\deg_x A$ *and* $\deg_x B$, *but* A *and* B *don't divide each other. Hence either* $y = 1$ *or* $p = 11$ *caused a problem but we don't know. For convenience we pick a new prime* $p = 13$, *evaluation points* $y = 1$ *and* $y = 2$ *and compute*

$$\begin{array}{rcccccccl}
g_{13,1} = \gcd(A(x,1), B(x,1)) = & \boxed{1} & x^2 & \boxed{+5} & x & \boxed{+8} & \mod 13 \\
g_{13,2} = \gcd(A(x,2), B(x,2)) = & \boxed{1} & x^2 & \boxed{+7} & x & \boxed{+8} & \mod 13 \\
\text{(Interpolate } y\text{)} & \Downarrow & & \Downarrow & & \Downarrow & \\
g_{13} = & \boxed{1} & x^2 & \boxed{+2y+3} & x & \boxed{+8} & \mod 13
\end{array}$$

Since g_{13} *does not divide* A *modulo* 13, *we need more points and we pick* $y = 3$.

$$\begin{array}{rcccccccl}
g_{13,1} = \gcd(A(x,1), B(x,1)) = & \boxed{1} & x^2 & \boxed{+5} & x & \boxed{+8} & \mod 13 \\
g_{13,2} = \gcd(A(x,2), B(x,2)) = & \boxed{1} & x^2 & \boxed{+7} & x & \boxed{+8} & \mod 13 \\
g_{13,3} = \gcd(A(x,3), B(x,3)) = & \boxed{1} & x^2 & \boxed{+6} & x & \boxed{+8} & \mod 13 \\
\text{(Interpolate } y\text{)} & \Downarrow & & \Downarrow & & \Downarrow & \\
g_{13} = & \boxed{1} & x^2 & \boxed{+5y^2} & x & \boxed{+8} & \mod 13
\end{array}$$

$$g_{13,3} = \gcd(A(x, 3), B(x, 3)) = x^2 + 6x + 8 \mod 13.$$

The symmetric representation of g_{13} *modulo* 13 *is*

$$g_{13} = x^2 + 5xy^2 - 5$$

which divides A and B in $\mathbb{Z}_{13}$ but does not divide A or B over $\mathbb{Z}$ hence we need more bivariate images. Let $p = 17$ be a new prime. Since $\deg_y g_{13} = 2$ we compute 3 univariate GCD images with evaluation points $1, 2, 3$ and interpolate y in the coefficients. Then we have

$$\begin{array}{rcccccccc}
g_{17,1} = \gcd(A(x,1), B(x,1)) = & \boxed{1} & x^2 & \boxed{+5} & x & \boxed{+16} & \mod 17 \\
g_{17,2} = \gcd(A(x,2), B(x,2)) = & \boxed{1} & x^2 & \boxed{+3} & x & \boxed{+16} & \mod 17 \\
g_{17,3} = \gcd(A(x,3), B(x,3)) = & \boxed{1} & x^2 & \boxed{+11} & x & \boxed{+16} & \mod 17 \\
\text{(Interpolate } y\text{)} & \Downarrow & & \Downarrow & & \Downarrow & \\
g_{17} = & \boxed{1} & x^2 & \boxed{+5y^2} & x & \boxed{+16} & \mod 17
\end{array}$$

Finally we apply the Chinese remaindering to g_{13}, g_{17} and obtain the symmetric representation

$$G = 5xy^2 + x^2 - 18 \mod 13 \cdot 17.$$

Since $G|A$ and $G|B$ over $\mathbb{Z}$, we conclude that $G = \gcd(A, B)$.

As we saw in the above example, the modular algorithm reduces a complex problem into a series of simpler problems which are easier to solve because the arithmetic is done in a smaller domain. But the trade-off is the information loss which may cause unexpected failure. Also, in Example 11, G is monic in x, the main variable. If not then one should also carefully handle the leading coefficient in each recursive level.

Let d bound the degree of each variable in the target GCD G and n be the number of variables. Brown's algorithm uses $O((d+1)^{n-1})$ points to densely interpolate G, for example, by Newton's method which assumes none of the possible terms is absent. However if the target GCD is sparse then Brown's algorithm is not efficient.

Example 1.12. *Suppose the GCD is $G = x_0^{1000} + x_1^{1000} + \cdots + x_4^{1000} + 1$. G has only 6 terms but each variable has degree 1000. With Brown's dense GCD algorithm we need at least 1001^4 points to recover G from univariate images in $\mathbb{Z}_p[x_0]$.*

In Example 1.12, G is a very sparse polynomial. Sparse polynomials, not typically this sparse, occur in many practical applications. e.g. Lewis determinants. Hence we are especially interested in designing GCD algorithms focusing on sparsity. Zippel's GCD algorithm is such an example. We should also mention that the GCD of two sparse polynomials A and B could be dense.

Example 1.13. *The following GCD has more terms than either input polynomial hence denser.*

$$\begin{aligned}
&\gcd(x^{18} - x^{14}y^4 - x^{12}y^6 + x^6y^{12} + x^4y^{14} - y^{18}, x^{30} + y^{30}) \\
&= x^{16} + x^{14}y^2 - x^{10}y^6 - x^8y^8 - x^6y^{10} + x^2y^{14} + y^{16}.
\end{aligned}$$

1.6 Zippel's algorithm

Zippel presented a sparse interpolation algorithm in 1979 [Zippel, 1979b]. It was developed to solve GCD problems. Nowadays Zippel's GCD algorithm denominates the modern computer algebra systems for multivariate polynomial GCD computation. For $A, B \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$, let $G = \gcd(A, B)$ and $\deg G = d$. Zippel's GCD algorithm reduces the number of evaluation points from Brown's $O((d+1)^n)$ to $O(ndt)$ where t is the number of non-zero terms in the target polynomial. The following example illustrates the sparse interpolation step in Zippel's GCD algorithm.

Example 1.14. *Let us compute the GCD of the following two multivariate polynomials in* $\mathbb{Z}[x, y, z]$.

$$A = x^3y + 6x^3 + 11x^2y^3 + 5x^2y^2z + 66x^2y^2 + 30x^2yz - 3y^3z - 18y^2z + 8y + 48,$$
$$B = x^4 + 11x^3y^2 + 5x^3yz - x^3 - 11x^2y^2 - 5x^2yz - 3xy^2z + 8x + 3y^2z - 8.$$

Let $p = 13$ *and assume we already have recursively computed*

$$g_{13} = \gcd(f_1, f_2) \mod p = x^3 + 11x^2y^2 + 5x^2yz + 10y^2z + 8.$$

Zippel's algorithm makes the following key assumption:

$$Supp(g_{13}) = Supp(g) = \{x^3, x^2y^2, x^2yz, y^2z, 1\}.$$

That is if the prime is large enough, the set of monomials in modular images is equal to the set of monomials in the true GCD. In this example for the demonstration purpose we only use small primes. With $Supp(g_{13})$ *we set the assumed form to be*

$$g_{form} = ax^3 + bx^2y^2 + cx^2yz + dy^2z + e.$$

We pick a new prime $p = 17$ *and the first evaluation point* $\beta_1 = (y = 1, z = 1)$ *uniformly at random then we evaluate*

$$g_{form}(x, \beta_1) = ax^3 + bx^2 + cx^2 + d + e, \tag{1.3}$$

and compute

$$g_{17,1} = \gcd(A(x, \beta_1), B(x, \beta_1)) = x^3 + 16x^2 + 5 \mod p \tag{1.4}$$

using the Euclidean algorithm in $\mathbb{Z}_p[x]$. *Let* $g_{form}(x, \beta_1) = g_{17,1}$. *Equating the coefficients of* x_i *in Equation* (1.3) *and Equation* (1.4) *we have three linear systems*

$$\{a = 1\}, \quad \{b + c = 16\}, \quad \{d + e = 5\}.$$

We need one more point to solve the systems. We pick $\beta_2 = (y = 2, z = 2)$ *uniformly at random and compute*

$$g_{form}(x, \beta_2) = ax^3 + 4bx^2 + 4cx^2 + 8d + e$$

and

$$g_{17,2} = \gcd(A(x, \beta_2), B(x, \beta_2)) = x^3 + 13x^2 + 1 \mod p.$$

Let $g_{form}(x, \beta_2) = g_{17,2}$*. Equating the coefficients of* x_i *we have*

$$\{a = 1\}, \quad \{4b + 4c = 13\}, \quad \{8d + e = 1\}.$$

We combine those six equations and obtain three linear systems

$$\{a = 1\}, \quad \{b + c = 16, 4b + 4c = 13\}, \quad \{d + e = 5, 8d + e = 1\}.$$

After solving the systems modulo p*, we have*

$$\{a = 1\}, \quad \{d = 14\}, \quad \{e = 8\},$$

but b *and* c *remain undermined because* β_2 *has* $y = z$*. In fact any point such that* $y = z$ *cannot be used because the coefficient of* x^2 *is* $by^2 + cyz$ *and we always get the same equation modulo* 17*. We choose another evaluation point* $\beta_3 = (y = 1, z = 2)$ *uniformly at random and compute*

$$g_{form}(x, \beta_3) = ax^3 + bx^2 + 2cx^2 + 2d + e,$$

and

$$g_{17,3} = \gcd(A(x, \beta_3), B(x, \beta_3)) = x^3 + 4x^2 + 2 \mod p.$$

Let $g_{form}(x, \beta_3) = g_{17,3}$*. Equating the coefficients of* x_i *we have*

$$\{a = 1\}, \quad \{b + 2c = 4\}, \quad \{2d + e = 2\}.$$

We combine those equations derived from points β_1, β_3 *and obtain three linear systems*

$$\{a = 1\}, \quad \{b + c = 16, b + 2c = 4\}, \quad \{d + e = 5, 2d + e = 2\}.$$

Solving the systems modulo p *gives the solutions*

$$\{a = 1\}, \quad \{b = 5, c = 1\}, \quad \{d = 14, e = 8\}.$$

Evaluating g_{form} at the above solutions gives

$$g_{17} = x^3 + 5x^2y^2 + x^2yz + 14y^2z + 8.$$

Zippel observed that if the evaluation points are chosen uniformly at random from a large set, then evaluating a non-zero polynomial at that point is rarely zero. Therefore if p is large enough then the assumed form g_{form} contains the complete support of the target GCD with high probability, that is $Supp(G) = Supp(g_{form})$. Zippel's algorithm interpolates the target polynomial by alternating from dense interpolation and sparse interpolation. Consequently the number of evaluation points is reduced.

In Example 1.14, with the assumed form, we only used 3 points to determine a GCD image with a new prime. In fact two points should be enough if we did not encounter that unlucky case. But Brown's algorithm requires at least $(\deg_y g_{13} + 1)(\deg_z g_{13} + 1) = 6$ points. The number of points required by Zippel's interpolation algorithm can be derived as follows. Let $f \in \mathbb{Z}_p[x_1, \ldots, x_n]$ where p is a large prime. Let $d_i = \deg_{x_i} f$ and $t_{1,\ldots,i}$ be an upper bound of the number of non-zero terms in $f(x_1, \ldots, x_i, x_{i+1} = \beta_i, \ldots, x_n = \beta_n)$ where $\beta_j \in \mathbb{Z}_p$ for $i + 1 \leq j \leq n$. To interpolate x_1, we need $d_1 + 1$ evaluation points for dense interpolation. To interpolate x_2, we need d_2 evaluation points together with $d_1 + 1$ points (used to interpolate x_1) which setup the assumed form. For each of d_2 points, only t_1 evaluation points are required to interpolate a new image in x_1 by Zippel's sparse method. We repeat the process to interpolate $x_3, x_4, \ldots$ and obtain the total number of evaluation points:

$$(d_1 + 1) + d_2t_1 + d_3t_{1,2} + d_nt_{1,2,\ldots,n-1}.$$

Let $d \geq d_i$ for $1 \leq i \leq n$ and $t \geq t_{1,2,\ldots,i}$ where $1 \leq i \leq n-1$. The total number of evaluation points in Zippel's sparse interpolation is in $O(ndt)$. For larger problems in many variables, the gain by the sparse interpolation is huge when be compared to Brown's dense method which requires $O(d^n)$ evaluation points.

The GCD computed in Example 1.14 is monic hence Zippel's GCD algorithm can be applied directly. For the non-monic GCD problem Zippel's GCD algorithm could run into a problem because univariate GCD images are not able to be scaled consistently.

Example 1.15. *Let us consider the GCD problem*

$$g = (y - 14)x^2 + (y^2 + 20)x = \gcd(A(x, y), B(x, y))$$

where $A, B \in \mathbb{Z}[x, y]$. Suppose we have determined

$$g_{13} \equiv (y + 12)x^2 + (y^2 + 7)x \mod 13.$$

The assumed form is

$$g_{form} = (ay + b)x^2 + (cy^2 + d)x.$$

Let us choose a new prime $p = 17$ *with evaluation points* $y = 1, 2$*, then we have*

$$ax^2 + bx^2 + cx + dx \mod 17 = g_{form}(x, 1) = g_{17}(x, 1) = x^2 + x \mod 17.$$

$$2ax^2 + bx^2 + 4cx + dx \mod 17 = g_{form}(x, 2) = g_{17}(x, 2) = x^2 + 15x \mod 17.$$

Note the Euclidean algorithm over $\mathbb{Z}_p[x]$ *always returns the monic GCD. We extract coefficients from above two equations to form two linear systems*

$$\{a + b = 1, 2a + b = 1\}, \quad \{c + d = 1, 4c + d = 15\}.$$

The solution to the systems modulo 17 *is*

$$\{a = 0, b = 1\}, \quad \{c = 16, d = 2\},$$

which is wrong as the leading term in $g_{form}(a = 0, b = 1, c = 16, d = 2)$ *vanishes.*

The univariate GCD images computed by the Euclidean algorithm over a field are set to be monic hence the information of the leading coefficient is lost if the target GCD is not monic. The univariate GCD image must be scaled by the leading coefficient evaluated at the same point in order to be usable. This is called the *normalization problem* which is to be discussed in Section 1.9.

1.7 Encarnacion's algorithm

Langemyr and McCallum successfully adapted Brown's modular GCD algorithm to univariate polynomial GCD computation over number fields. Their algorithm requires an *algebraic integer* extension. Moreover the GCD usually has coefficients in $\mathbb{Q}(\alpha)$ even though the inputs have coefficients in $\mathbb{Z}(\alpha)$, hence a denominator bound would be necessary but such a bound is generally too large. Encarnacion's algorithm not only generalizes Langemyr and McCallum's algorithm to *algebraic number* extensions but also uses rational number reconstruction to recover the rational coefficients in the target GCD hence a denominator bound is not required anymore. See Section 1.8 for more detail about rational number reconstruction.

In this section, we give an example to briefly review univariate polynomial GCD computation over $\mathbb{Q}(\alpha)$ where α is an algebraic number. We focus on Encarnacion's algorithm because it is the fastest algorithm for a simple extension.

Example 1.16. *Suppose we want to find* $G = \gcd(A, B)$ *where*

$$A = (8x^4\sqrt{2} + 9x\sqrt{2} + 7x)(-6x^3\sqrt{2} - 5x^2) \quad \textit{and} \quad B = (8x^4\sqrt{2} + 9x\sqrt{2} + 7x)(-10x^2 - 36).$$

We first choose the prime $p = 53$*. In order to compute* $\gcd(A, B) \mod p$*, we use the Euclidean algorithm with coefficient over a ring as* $\mathbb{Q}(\sqrt{2}) \mod 53$ *which, although not a field, is a finite ring. Let*

$$\begin{aligned} r_1 &= A \mod p = 10x^7 + 13\sqrt{2}x^6 + (11\sqrt{2} + 51)x^4 + (8\sqrt{2} + 18)x^3, \\ r_2 &= B \mod p = 26\sqrt{2}x^6 + 30\sqrt{2}x^4 + (16\sqrt{2} + 36)x^3 + (47\sqrt{2} + 13)x. \end{aligned}$$

In order to proceed with the Euclidean algorithm we first make r_2 *monic which is*

$$monic(r_2) = x^6 + 46x^4 + (17\sqrt{2} + 21)x^3 + (40\sqrt{2} + 12)x$$

where $monic(r_2) = LC(r_2)^{-1}r_2$*. The remainder of* r_1 *and* $monic(r_2)$ *is*

$$r_3 = 17x^5 + 38\sqrt{2}x^4 + (24\sqrt{2} + 39)x^2 + (3\sqrt{2} + 20)x,$$

and

$$monic(r_3) = x^5 + 49\sqrt{2}x^4 + (17\sqrt{2} + 21)x^2 + (22\sqrt{2} + 23)x.$$

The remainder of $monic(r_2)$ *and* $monic(r_3)$ *is*

$$r_4 = 25x^4 + (\sqrt{2} + 48)x \quad \textit{and} \quad monic(r_4) = x^4 + (17\sqrt{2} + 21)x.$$

The remainder of $monic(r_3)$ *and* $monic(r_4)$ *is* 0 *hence* $G_{53} = \gcd(A, B) \mod p = monic(r_4)$*. Next we choose the prime* $p = 59$ *and obtain* $G_{59} = x^4 + (41\sqrt{2} + 38)x$*. Now we treat* $\sqrt{2}$ *as a variable and apply the Chinese remaindering to coefficients of* G_{53} *and* G_{59} *with respect to variables* x *and* $\sqrt{2}$ *and get* $g = x^4 + 2932\sqrt{2}x + 392x$*. We apply the rational number reconstruction to the integer coefficients of* g *with moduli* $53 \cdot 59$ *and obtain* $G = x^4 + (\frac{7}{16}\sqrt{2} + \frac{9}{8})x$*. Since* G *divides* A *and* B*,* $G = \gcd(A, B)$*.*

The new technical difficulty for the modular GCD algorithm over number fields is the presence of zero divisors. As we saw in above example, in order to make divisors (remainders) monic we have to calculate the inverse of the leading coefficient of the remainder over a finite ring which could fail if the leading coefficient is a zero divisor. In fact any inversion in a finite ring in this algorithm could fail due to zero divisors. For example, let α be an algebraic number so that $\alpha^3 - 2 = 0$. If the leading coefficient of some remainder is $\alpha + 2$ and we use the prime 5 then we encounter a zero divisor when inverting $\alpha + 2 \mod 5$ because $\alpha^3 - 2 \mod 5 = (\alpha + 2)(\alpha^2 + 3\alpha + 4)$. We will return to the problem of zero divisors in Section 8.5.

1.8 Rational number reconstruction

In this section we review rational number reconstruction. One contribution by Encarnacion [Encarnacion, 1995] is to use rational number reconstruction [Wang, 1981] to recover the coefficients of GCD. Let $a/b \in \mathbb{Q}$ with $\gcd(a, b) = 1$, $m \in \mathbb{Z}$ with $\gcd(b, m) = 1$ and $c \equiv a/b \mod m$. The rational number reconstruction problem is to find a and b given c and m are known. Recall that in the extended Euclidean algorithm, every remainder r_i can be written as the linear combination of m and c:

$$s_i m + t_i c = r_i$$

for some integers s_i and t_i. Therefore $t_i c \equiv r_i \mod m$. If $\gcd(t_i, m) = 1$, then $c \equiv \frac{r_i}{t_i} \mod m$. Hence we have a set of possible solutions $S = \{\frac{r_i}{t_i}\}_i$.

Example 1.17. *Let $c = 11$ and $m = 19$. We run the extended Euclidean algorithm on m and c and track $\frac{r_i}{t_i}$, then get the following sequence*

$$S = \{-\frac{8}{1}, \frac{3}{2}, -\frac{2}{5}, \frac{1}{7}\}.$$

Every number in S is equal to 11 modulo 19.

Wang [Wang, 1981, Section 4] shows that $\frac{a}{b}$ can be uniquely determined from S if $|a| < \sqrt{m/2}$ and $|b| < \sqrt{m/2}$, that is there exists a unique index i in the Euclidean algorithm such that $\frac{r_i}{t_i} = \frac{a}{b}$. In other words, $m > 2|a||b|$ ensures $\frac{a}{b} \in S$. In fact the first i such that $r_i \leq |a|$ is the index. Wang's algorithm assumes the sizes of numerator and denominator have equal growth rate as m increases. Wang's rational number reconstruction algorithm outputs $\frac{a}{b}$ for all $m > 2\max(|a|, |b|)^2$. In practice, the denominator b may be much smaller than the numerator a. For example, polynomial computations over a number field $\mathbb{Q}(\alpha)$. Hence $|b| < \sqrt{m/2}$ may force us to use a too large m.

In our problem, we compute a series of modular GCD images modulo $p_1, p_2, \ldots$ and use the Chinese remaindering to combine the moduli so that $p_1 \cdot p_2 \cdots p_k \geq m$, then apply the rational number reconstruction to recover the rational coefficients of the target GCD. If α is an algebraic integer whose minimal polynomial is monic over $\mathbb{Z}$, Wang's algorithm may use up to twice as many primes as are necessary hence requires us to compute more modular GCD images which is the most expensive part in our algorithm. In order to overcome this issue, Monagan [Monagan, 2004] presented the *maximal quotient rational reconstruction* or MQRR. It is based on the observation that if q_{i+1} is the maximum quotient in the extended Euclidean algorithm then $\frac{r_i}{t_i}$ is probably $\frac{a}{b}$.

Example 1.18. *Consider t_i and q_i in the extended Euclidean algorithm with inputs $c = 22234$ and $m = 99991$.*

	t_i	r_i	q_i
1	*0*	*m*	
2	*1*	*c*	*0*
3	*-4*	*11055*	*4*
4	*9*	*124*	*2*
5	*-805*	*19*	*89*
6	*4839*	*10*	*6*
7	*-5644*	*9*	*1*
8	*10483*	*1*	*1*
9	*-99991*	*0*	*9*

We observe that the row 5 *in above table has the maximum quotient* 89*, therefore row* 4 *should correspond to the desired rational number which is* $\frac{r_4}{t_4} = \frac{124}{9} = 22234 \mod 99991$*. See [Monagan, 2004] Lemma 1 for a good reason.*

It can be shown that there is only one maximal quotient if m is large. For the worst case MQRR yields no improvement over Wang's algorithm. But in the average case MQRR determines the result when m is a few bit longer than $2|ab|$ with high probability. There is a new parameter T for the input so that MQRR returns $\frac{r_i}{t_i}$ and q_{i+1} is maximal, $q_{i+1} > T$. For reasonable choices of T for real problems, see [Monagan, 2004, Section 2]. If we use the classic Euclidean algorithm, the complexity of the rational number reconstruction is $O(\log^2 m)$. Suppose $f \in \mathbb{Q}[x]$ and $g = f \mod m$. If m is not large enough, there is a significant probability that $F \neq f$ where F is the result of successfully performing the rational number reconstruction on all coefficients of g without failure. If this happens we try a larger modulus m. For sufficiently large m, $F = f$.

1.9 The normalization problem

The normalization problem seems to be common in every major multivariate polynomial GCD algorithm as we saw. Suppose now we have a modular algorithm to compute the GCD of multivariate polynomials in $\mathbb{Z}[x_0, x_1, \ldots, x_n]$. Let $LC(G)$ be the leading coefficient of $G = \gcd(A, B)$ with respect to the main variable x_0. One solution to the normalization problem is to multiply the monic GCD images by the image of a polynomial called the *scaling factor* which is divisible by $LC(G)$. Naively, $LC(A)$, $LC(B)$ and $\Gamma = \gcd(LC(A), LC(B))$ all are eligible to be the scaling factor. Zippel implemented his GCD algorithm with Γ as the scaling factor in Macsyma. Let $\Delta \in \mathbb{Z}[x_1, \ldots, x_n]$ so that $\Gamma = LC(G)\Delta$. If Γ is the scaling factor then the polynomial GCD we interpolate is $H = \Delta G$. Therefore, in the last step of the GCD algorithm the content of H with respect to x_0 must be removed to get the correct G. If Δ consists of more than one term, the sparsity is changed and it may lead us to interpolate a much larger polynomial and with a higher cost to remove the content.

Example 1.19. *Consider the GCD problem in* $\mathbb{Z}[x, y]$*. Let*

$$A = ((y^3 + y^2 + y + 1)x + 1)(x + y + 1) \quad \textit{and} \quad B = ((y^3 + y^2 + y + 1)x + 2)(x + y + 1).$$

Then $\Gamma = y^3 + y^2 + y + 1$*. But* $\gcd(A, B) = x + y + 1$ *hence its leading coefficient is* 1*. If we use* Γ *as the scaling factor then we actually compute* $(y^3 + y^2 + y + 1)(x + y + 1)$ *and the polynomial content* $y^3 + y^2 + y + 1$ *must be removed at the end.*

In the EEZ-GCD algorithm [Wang, 1980] Wang designs a clever leading coefficient predetermination algorithm GCDLC which heuristically removes Δ from the scaling factor and determines the leading coefficient of the target GCD up to a scalar multiple, hence it does not change the sparsity of the polynomial to be interpolated. However multivariate polynomial factorization is required in GCDLC and giant integer GCD calculation is often seen in GCDLC if the total degree of Γ is high or n is large. We demonstrate Wang's leading coefficient algorithm with the following example.

Example 1.20. *Let* $A = \bar{A}G, B = \bar{B}G \in \mathbb{Z}[x, y, z, u, v]$ *where* $G = \gcd(A, B)$ *and* $\bar{A}, \bar{B}$ *are cofactors.*

$$\bar{A} = (4uy - v)(2z - 3v)x + 2, \qquad \bar{B} = (2z - 3v)x + 2,$$

$$G = 56(-5u^2 + 11z)(3y - 7z)(7y + 4v)^2x + 1.$$

Let $\Gamma = \gcd(LC(A), LC(B))$*. We factor* Γ *to obtain*

$$\Gamma = 56(3y - 7z)(3v - 2z)(7y + 4v)^2(5u^2 - 11z).$$

We construct a factor list

$$L = [56, 3v - 2z, 7y + 4v, 5u^2 - 11z, 3y - 7z]$$

and pick a random evaluation point

$$E = [y = 7926, \quad z = 8057, \quad v = 5, \quad u = 3002].$$

After evaluating the factor list at the point E*, we have*

$$LE = [56, \quad -16099, \quad 55502, \quad 44971393, \quad -32621.]$$

We compute the GCD of A *and* B *evaluated at* E *and get*

$$g(x, E) = \gcd(A(x, E), B(x, E)) = 25306897355522183857187 2x + 1.$$

The integer content of $g(x, E)$ *is 1 hence* $LE_1 = 56 \cdot 1 = 56$*. Now we call AlgorithmN (see [Wang, 1980]) with input* LE*. AlgorithmN computes a list of divisors* D *so that the size of* D

is equal to the size of LE and D_i divides LE_i but D_i does not divide LE_j for $1 \le j \le i-1$. If AlgorithmN fails to find such a divisor list, then the algorithm evaluates L at a new point and calls AlgorithmN again until such a list is found. AlgorithmN rarely fails with a random and large evaluation. In this example, AlgorithmN returns

$$D = [56, \quad 16099, \quad 27751, \quad 44971393, \quad 32621].$$

Let $LG = LC(g(x, E)) = 253068973555221838571872$. We use repeated trial division by D_i to determine factors in the leading coefficient and their multiplicities.

1. 32621^1 *divides LG but* 32621^2 *does not, so the factor* $3y-7z$ *is in the leading coefficient of g. Set* $LG = LG/32621$.
2. 44971393^1 *divides LG but* 44971393^2 *does not, so the factor* $5u^2-11z$ *is in the leading coefficient of g. Set* $LG = LG/44971393$.
3. 27751^2 *divides LG but* 27751^3 *does not, so the factor* $(7y + 4v)^2$ *is in the leading coefficient of g. Set* $LG = LG/27751^2$.
4. 16099 *does not divide LG, so* $3v-2z$ *must be factor of leading coefficients of cofactors.*
5. 56 *divides LG but* 56^2 *does not, so the factor* 56 *is in the leading coefficient of g. We run out of D and stop.*

Hence the leading coefficient of g is

$$56(3y - 7z)(5u^2 - 11z)(7y + 4v)^2.$$

In Example 1.20 we use a 4 digit evaluation point for 4 variables and the total degree in the leading coefficient of the GCD is 5. The integers in the computation are already large compared to the magnitude of the coefficients of input polynomials. For problems with more variables in higher degree, the size of the integers could blow up easily.

In 2005 Kleine, Monagan and Wittkopf [de Kleine et al., 2005] presented a variant of Zippel's GCD algorithm called LINZIP which automatically handles the normalization problem. The idea is to treat multipliers m_i(scaling factors) for the monic univariate GCD images as unknowns. The trade-off is that one needs to use more equations as needed to account for the m_i. Consequently more univariate GCD images are required to form a larger linear system. It is the default multivariate GCD algorithm over $\mathbb{Z}$ currently used by Maple.

Let $A, B \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$ and $g = \gcd(A, B)$. Suppose the assumed form is

$$g_{form} = C_1 x_0^{e_1} + C_2 x_0^{e_2} + \cdots + C_s x_0^{e_s},$$

where $C_i \in \mathbb{Z}[x_1, \ldots, x_n]$, e_i is non-negative integer for $1 \le i \le s$ and $e_i \ne e_j$ if $i \ne j$. Let t_i denote the number of terms in C_i for $1 \le i \le s$ and $t_{max} \ge t_i$ for $1 \le i \le s$. In LINZIP one

univariate GCD image introduces s new equations and one new unknown (the multiplier). Hence the total number of univariate GCD images required to solve the linear system is

$$\max(\lceil \frac{\sum_{i=1}^{s} t_i}{s-1} \rceil, t_{max}).$$

The complexity of LINZIP is $O(t_1^3 + t_2^3 + \cdots + t_s^3)$. Javadi [Javadi, 2008] improved LINZIP by carefully choosing where to scale and using specific evaluation points so that Zippel's quadratic Vandermonde linear system solving algorithm can be applied. Practically we set the first multiplier m_1 to be 1. We demonstrate this algorithm by redoing Example 1.15.

Example 1.21. *Consider the GCD problem*

$$G = (y-14)x^2 + (y^2+20)x = \gcd(A(x,y), B(x,y))$$

where $A, B \in \mathbb{Z}[x,y]$. *Suppose we already have*

$$g_{13} = (y+12)x^2 + (y^2+7)x \mod 13.$$

Hence the assumed form for G *is*

$$g_{form} = (ay+b)x^2 + (cy^2+d)x.$$

Let us choose a new prime $p = 17$ *with evaluation points* $y = 1, 2$ *and* 3. *We have equations*

$$\begin{aligned} ax^2 + bx^2 + cx + dx &= m_1(x^2+x) = 1(x^2+x), \\ 2ax^2 + bx^2 + 4cx + dx &= m_2(x^2+15x), \\ 3ax^2 + bx^2 + 9cx + dx &= m_3(x^2+2x). \end{aligned}$$

We form a linear system by equating the coefficients and obtain

$$a+b = m_1,\ 2a+b = m_2,\ 3a+b = m_3,\ c+d = 1,\ 4c+d = 15m_2,\ 9c+d = 2m_3.$$

By solving the system with $m_1 = 1$, *we have*

$$a = 13, \quad b = 5, \quad c = 13, \quad d = 5, \quad m_2 = 14, \quad m_3 = 10.$$

Hence $g_{17} = 13x^2y + 5xy^2 + 13x^2 + 5x \mod 17$. *If we make* g_{17} *monic, then* $g_{17} \equiv g \mod 17$.

LINZIP may encounter an *unlucky content* which results in an undetermined linear system no matter how many univariate GCD images are computed. But the unlucky content is rare and the solution is simply to use a new prime.

Recently, Tang, Li and Zeng [Tang et al., 2018] presented a method to scale the univariate GCD images correctly by homogenizing. For $A, B \in K[x_0, \ldots, x_1]$ where K is a field they introduces the homogenizing variable z to A and B by computing

$$A_h = A(z \cdot x_0, z \cdot x_1, \ldots, z \cdot x_n) = \sum A_i(x_0, \ldots, x_n) z^i \text{ and}$$
$$B_h = B(z \cdot x_0, z \cdot x_1, \ldots, z \cdot x_n) = \sum B_i(x_0, \ldots, x_n) z^i,$$

where A_i and B_i are homogeneous polynomials. Let z be the main variable. Then

$$g_h = \gcd(A_h, B_h) = \sum_{i=k+\tau}^{d+\tau} G_{i-\tau}(x_0, \ldots, x_n) z^i.$$

But for an evaluation point $\alpha \in K^{n+1}$, $\gcd(A_h(z, \alpha), B_h(z, \alpha))$ is monic in z we need to determine the correct scaling factor. Let $(\sigma_0, \ldots, \sigma_n) \in K^{n+1}$ be a random point. They solved this issue by computing

$$A_{hs} = A(z \cdot x_0 + \sigma_0, z \cdot x_1 + \sigma_1, \ldots, z \cdot x_n + \sigma_n) = \sum As_i(x_0, \ldots, x_n) z^i \text{ and}$$
$$B_{hs} = B(z \cdot x_0 + \sigma_0, z \cdot x_1 + \sigma_1, \ldots, z \cdot x_n + \sigma_n) = \sum Bs_i(x_0, \ldots, x_n) z^i$$

Since A_{hs} and B_{hs} has non-zero constant terms with high probability we have $g_{hs} = \gcd(A_{hs}, B_{hs}) = \sum_{i=1}^{d} Gs_i(x_0, \ldots, x_n) z^i + 1$. Since there is a constant term in g_{hs}, we can scale the images of $\gcd(A_{hs}(z, \alpha), B_{hs}(z, \alpha))$ correctly to obtain g_{hs}. Since $Gs_d(x_0, \ldots, x_n) = cG_d(x_0, \ldots, x_n)$ for $c \in K \backslash \{0\}$, we can use the leading coefficient of $\gcd(A_{hs}(z, \alpha), B_{hs}(z, \alpha))$ to scale the images of $\gcd(A_h(z, \alpha), B_h(z, \alpha))$ to obtain $\gcd(A_h, B_h)$. The GCD of A and B is $G = \gcd(A, B) = \sum_{i=k}^{d} G_i(x_0, \ldots, x_n)$.

In our algorithm, we have the normalization problem too. However even the GCD of the leading coefficients of the input polynomials could lead us to an infinite loop as we always fail to meet some termination condition. This will be discussed in the next chapter.

1.10 Thesis outline

We first quickly review some standard definitions so that readers can understand the thesis outline better. The formal definitions are given when needed in the analysis later. Suppose $A, B, f \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$ and $G = \gcd(A, B) = \sum_{i=0}^{d_G} c_i x_0^i$. Let $LC(f)$ be the leading coefficient of f with respect to main variable x_0, $d = \max\{\max_{i=1}^{n}(\deg_{x_i} A, \deg_{x_i} B)\}$, $\#f$ be the number of non-zero terms in f and $t = \max_{i=0}^{d_G}\{\#c_i\}$ be a term bound. Let p be a prime and $\beta \in \mathbb{Z}_p^n$ be an evaluation point. If $LC(A(x_0, \beta)) \equiv 0 \mod p$ or $LC(B(x_0, \beta)) \equiv 0 \mod p$ then β is called *bad*. If β is not bad and $\deg_{x_0} \gcd(A(x_0, \beta), B(x_0, \beta)) > \deg_{x_0} G$ then β is called *unlucky*. Let $\Gamma = \gcd(LC(A), LC(B))$. Then $\Gamma = \Delta LC(G)$, hence $\Delta = \gcd(LC(\bar{A}), LC(\bar{B}))$ is contributed by the cofactors of A and B. Let $H = \Delta G$.

In Chapter 2, we first give a review for Ben-Or/Tiwari sparse interpolation. The evaluation points presented in Ben-Or/Tiwari interpolation are prime numbers. If we impose the modular arithmetic naively on Ben-Or/Tiwari interpolation by choosing a prime so that all monomials in the target polynomial can be recovered exactly then the prime could be easily greater than 2^{64} hence the algorithm would require multi-precision arithmetic instead of machine arithmetic. Therefore we propose to use the Ben-Or/Tiwari algorithm with the discrete logarithm method, which uses the powers of primitive elements in the cyclic group $\mathbb{Z}_p^*$ as evaluation points and computes discrete logarithms in $\mathbb{Z}_p^*$. The advantage over the original Ben-Or/Tiwari algorithm is that the size of primes only depends on the degrees of H in $x_1, \ldots, x_n$ hence is manageable. We also analyze the running time of this algorithm. Then we provide an example of the new GCD algorithm to illustrate the computation procedure. Another problem in Ben-Or/Tiwari interpolation is that t, the number of terms in the target GCD, is not given. Hence a termination mechanism to determine t must be given. This has been discussed in [Ben-Or and Tiwari, 1988, E Kaltofen and Lobo, 2000]. We fill in more details about this topic to convince reader why it works. To reduce the probability of encountering unlucky evaluation points, the prime may need to be larger than the theoretical bound. Our modification for the discrete logarithm sequence increases the size of p which negates much of its advantage. This leads us to consider using a Kronecker substitution which is the main topic of Chapter 3.

In Chapter 3 we use a Kronecker substitution of the form

$$K_r(f(x_0, x_1, \ldots, x_n)) = f(x, y, y^{r_1}, \ldots, y^{r_1 \cdots r_{n-1}})$$

to map a multivariate computation to a bivariate computation, where r_i are non-negative integers for $1 \leq i \leq n-1$. Some Kronecker substitutions result in all evaluation points being unlucky. Those substitutions are called unlucky and can not be used. We show that there are only finitely many of them and how to detect them so that a larger Kronecker substitution may be tried. If a Kronecker substitution is not unlucky there can still be many unlucky evaluation points because the degree of $K_r(A)$ and $K_r(B)$ in y is exponential in n and higher degree implies possibly unlucky evaluation points. In order to avoid unlucky evaluation points one may simply choose the prime $p \gg \max(\deg_y(K_r(A)), \deg_y(K_r(B)))$, which is the strategy of the "simplified" version of our GCD algorithm in chapter 4. But if p is not a machine prime then we need to use multi-precision arithmetic which will significantly increase the cost of all modular arithmetic in $\mathbb{Z}_p$. However, it is well known that unlucky evaluation points are rare. In Theorem 3.3 we show that the expected number is 1 over all inputs.

In Chapter 4 we assemble a "Simplified Algorithm" which is a Las-Vegas GCD algorithm. It first applies a Kronecker substitution to map the GCD computation into $\mathbb{Z}[x, y]$. Then it chooses p uniformly at random from a large set of smooth primes and computes $H = G\Delta$

mod p using the Ben-Or/Tiwari algorithm to interpolate y. The algorithm then uses further primes and the Chinese remaindering to recover the integer coefficients in H and applies the inverse Kronecker substitution to obtain the final result in $\mathbb{Z}[x_0, x_1, \ldots, x_n]$. The algorithm chooses a Kronecker substitution large enough to be a priori not unlucky and also assumes a term bound for H is given. These assumptions lead to a much simpler algorithm.

In Chapter 5, we relax the term bound requirement and first try a Kronecker substitution just large enough to recover H, that is with $r_i \geq \deg_{x_i} H + 1$. The Kronecker substitution could be unlucky because the cofactors may contribute factors to H after substitution. We also don't have the term bound hence the Berlekamp-Massey algorithm may terminate early and output a wrong feedback polynomial from which we could obtain a set of wrong monomials. Those concerns complicate the GCD algorithm significantly. We present a heuristic GCD algorithm which we can prove always terminates and outputs the correct GCD. The heuristic algorithm will usually be much faster than the simplified algorithm but it can, in theory, fail several times before it finds a good Kronecker substitution K_r, a correct t, a sufficiently large prime p, and a sequence of evaluation points which are all good.

In Chapter 6, some implementation techniques and optimization options will be discussed. We notice that most of the time our algorithm is in evaluation. For larger GCD problems, see the bottom of Table 7.2, almost 99% running time contributes to evaluation. Therefore improving evaluations benefits the whole algorithm. Let $s = \#A + \#B$ and T be the number of evaluation points. We present two enhanced evaluation algorithms. One is Tuncer and Monagan's matrix evaluation method which costs $O(nd + ns + sT)$ multiplications to evaluate T points. Another one is Monagan and Wong [Monagan and Wong, 2017]'s fast parallel multi-point evaluation algorithm which further reduces the number of multiplications to $O(nd + ns + s\log^2 T)$ which effectively reduces the cost of evaluation in a larger problem, see [Monagan and Wong, 2017, Figure 6]. Another approach is to reduce t so fewer evaluation points are used. One way to reduce t is to evaluate the inputs A and B to bivariate polynomials instead of univariate ones and then compute their bivariate GCD. For example, let $G = x^2 + (z^2y^2 + z^2y + z + y) = x^2 + z^2y^2 + (z^2 + 1)y + z$. If we consider coefficients with respect to x, then $t = 4$. If we consider coefficients with respect to x and y, then $t = 2$. Another method is Homogenization. This is especially useful if A or B has a constant term. The main purpose of homogenization is to eliminate the normalization problem, which generally reduces t because $\Delta = 1$.

In Chapter 7, we compare our new algorithm with the C implementations of Zippel's GCD algorithm in Maple and Magma. The timing results are very promising. For our benchmark problem, Maple takes 22,111 seconds, Magma takes 1,611 seconds. Our new algorithm takes 48.17 seconds on 1 core and 4.67 seconds on 16 cores. If we compute bivariate GCD images in the base case, then our new algorithm takes 7.614 seconds on 1 core and 0.685 seconds on 16 cores.

In Chapter 8, we present a multivariate polynomial GCD algorithm over a number field $\mathbb{Q}(\alpha)$ where α is an algebraic number. The new algorithm is very similar to the "simplified" version GCD algorithm presented in Chapter 4. But the coefficients sit in a finite ring $\mathbb{Q}(\alpha) \mod p = \mathbb{Z}_p(\alpha)$ which may have zero divisors. In this algorithm the term bound is assumed to be given and the Kronecker substitution is chosen to be large enough so that it is always *good*. See Section 4.1 for the definition of "good". The presence of zero divisors and the operations in $\mathbb{Z}_p(\alpha)$ significantly complicate the analysis. For example, in order to get a bound for the number of zero divisors in $\mathbb{Z}_p(\alpha)$ we have to use the subresultant PRS method for GCD in $\mathbb{Z}_p(\alpha)[x]$. This GCD algorithm can be modified to a faster version by the identical approach presented in Chapter 5, which we don't discuss in this thesis.

In Chapter 9, we discuss some possible improvements of our GCD algorithms.

Chapter 2

Using Ben-Or Tiwari interpolation

In this chapter, we go through the detail of the Ben-Or/Tiwari interpolation and the discrete logarithm method. We also discuss that how to apply them to polynomial GCD problem and the difficulties we may encounter.

Let $A = G\bar{A} = \sum_{i=0}^{d_A} a_i x_0^i$, $B = G\bar{B} = \sum_{i=0}^{d_B} b_i x_0^i$ and $G = \gcd(A, B) = \sum_{i=0}^{d_G} c_i x_0^i$ where a_i, b_i and c_i are in $\mathbb{Z}[x_1, \ldots, x_n]$. $\bar{A}$ and $\bar{B}$ are *cofactors* of A and B respectively. Our GCD algorithm first computes and removes contents, that is to compute A and B such that $\mathrm{cont}(A, x_0) = \gcd(a_i) = 1$ and $\mathrm{cont}(B, x_0) = \gcd(b_i) = 1$. These GCD computations are in $\mathbb{Z}[x_1, x_2, \ldots, x_n]$ and may be computed recursively. We also assume that $d_A > 0$ and $d_B > 0$. If $d_A = 0$ we use $\gcd(A, B) = \gcd(a_0, B) = \gcd(a_0, \mathrm{cont}(B, x_0))$ to perform a GCD computation in one less variable.

Let $\#A$ denote the number of terms in A and let $\mathrm{Supp}(A)$ denote the support of A which is the set of monomials appearing in A. Let $\mathrm{LC}(A)$ denote the leading coefficient of A taken in x_0. Let $\Gamma = \gcd(LC(A), LC(B)) = \gcd(a_{d_A}, b_{d_B})$. Since $LC(G)|LC(A)$ and $\mathrm{LC}(G)|LC(B)$, it must be that $\mathrm{LC}(G)|\Gamma$. Therefore $\Gamma = LC(G)\Delta$ for some polynomial $\Delta \in \mathbb{Z}[x_1, \ldots, x_n]$.

Example 2.1. *If* $G = x_1x_0^2 + x_2x_0 + 3$, $\bar{A} = (x_2 - x_1)x_0 + x_2$ *and* $\bar{B} = (x_2 - x_1)x_0 + x_1 + 2$, *then we have* $\#G = 3$, $\mathrm{Supp}(G) = \{x_1x_0^2, x_2x_0, 1\}$, $\mathrm{LC}(G) = x_1$, $\Gamma = x_1(x_2 - x_1)$, *and* $\Delta = x_2 - x_1$.

We provide an overview of the GCD algorithm. Let $H = \Delta \times G$ and $h_i = \Delta \times c_i$ so that $H = \sum_{i=0}^{d_G} h_i x_0^i$. Our algorithm will compute H not G. After computing H, it then computes $\mathrm{cont}(H, x_0) = \gcd(h_i) = \Delta$ and divides H by Δ to obtain G. We compute H modulo a sequence of primes $p_1, p_2, \ldots$, and recover the integer coefficients of H using Chinese remainder theorem. The use of Chinese remaindering is standard. Details may be found in [Brown, 1971, Geddes et al., 1992]. Let H_1 be the result of computing $H \mod p_1$. For the remaining primes we use the sparse interpolation approach of Zippel [Zippel, 1979b] which assumes $\mathrm{Supp}(H_1) = \mathrm{Supp}(H)$. Let us focus on the computation of $H \mod p_1$.

Let $p = p_1$. To compute H mod p the algorithm will pick a sequence of points $\beta_1, \beta_2, \ldots$ from $\mathbb{Z}_p^n$, compute monic images

$$g_j := \gcd(A(x_0, \beta_j), B(x_0, \beta_j)) \in \mathbb{Z}_p[x_0]$$

of G, in parallel, then multiply g_j by the scalar $\Gamma(\beta_j) \in \mathbb{Z}_p$. Because the scaled image $\Gamma(\beta_j) \times g_j(x_0)$ is an image of H, we can use polynomial interpolation to interpolate each coefficient $h_i(x_1, \ldots, x_n)$ of H from the coefficients of the scaled images.

Let $t = \max_{i=0}^{d_G} \#h_i$. The parameter t measures the sparsity of H. Let $d = \max_{i=1}^{n} \deg_{x_i} H$ and $D = \max_{i=0}^{d_G} \deg h_i$. Note that $\deg h_i$ is the total degree of h_i. The cost of sparse polynomial interpolation algorithms is determined mainly by the number of points $\beta_1, \beta_2, \ldots$ needed and the size of the prime p needed. These depend on t, d, D and n. Table 1 below presents data for several sparse interpolation algorithms.

To get a sense for how large the prime needs to be for the different algorithms in Table 2.1, we include data for the following **benchmark problem:** Let $G, \bar{A}, \bar{B}$ have nine variables ($n = 8$), degree $d = 20$ in each variable, and total degree $D = 60$ (to better reflect real problems). Let G have 10,000 terms with $t = 1000$. Let $\bar{A}$ and $\bar{B}$ have 100 terms so that $A = G\bar{A}$ and $B = G\bar{B}$ have about one million terms.

	#points	size of p	benchmark
Zippel [1979]	$O(ndt)$	$p > 2nd^2t^2$	$= 6.4 \times 10^9$
BenOr/Tiwari [1988]	$O(t)$	$p > p_n^D$	$= 5.3 \times 10^{77}$
Monagan/Javadi [2010]	$O(nt)$	$p > nDt^2$	$= 4.8 \times 10^8$
Discrete Logs	$O(t)$	$p > (d+1)^n$	$= 3.7 \times 10^{10}$

Table 2.1: Some sparse interpolation algorithms

Remark 2.1. Kaltofen and Lee showed in [E Kaltofen and Lobo, 2000] how to modify Zippel's algorithm so that it will work effectively for primes much smaller than $2nd^2t^2$.

The Ben-Or/Tiwari algorithm [Ben-Or and Tiwari, 1988] is deterministic. The primary disadvantage of the Ben-Or/Tiwari algorithm is the size of the prime. In [Javadi and Monagan, 2010] Javadi and Monagan modified the Ben-Or/Tiwari algorithm to work for a prime of size $O(nDt^2)$ bits but require $O(nt)$ points. With experiments we find out that evaluations dominate the computational cost of the algorithm. Therefore we try to use as few evaluation points as possible. The Ben-Or/Tiwari algorithm with discrete logarithm method using $O(t)$ points achieves this goal best.

2.1 Some notations and results

The proofs in the thesis make use of properties of the Sylvester polynomial resultant, the Schwartz-Zippel Lemma and require bounds for the size of the integer coefficients appearing

in certain polynomials. We state these results in this section for later use. First we introduce some notation.

Let $f = \sum_{i=1}^{t} a_i M_i$ where $a_i \in \mathbb{Z}$, $a_i \neq 0$, $t \geq 0$ and M_i be a monomial in n variables $x_1, x_2, \ldots, x_n$. We denote the total degree of f by $\deg f$, the degree of f in x_i by $\deg_{x_i} f$, and the number of terms of f by $\#f$. We need to bound the size of the integer coefficients of certain polynomials. For this purpose let $\|f\|_1 = \sum_{i=1}^{t} |a_i|$ be the one-norm of f and $\|f\| = \max_{i=1}^{t} |a_i|$ be the height of f. For a prime p, let ϕ_p denote the modular map given by $\phi_p(f) = f \mod p$.

Lemma 2.1. *(Schwartz-Zippel [Schwartz, 1980, Zippel, 1979b]). Let F be a field and $f \in F[x_1, x_2, \ldots, x_n]$ be a non-zero polynomial with total degree d. Let $S \subset F$. If β is chosen uniformly at random from S^n then* $\text{Prob}[f(\beta) = 0] \leq \frac{d}{|S|}$. *Hence if $R = \{\beta | f(\beta) = 0\}$ then $|R| \leq d|S|^{n-1}$.*

Lemma 2.2. *(Gelfond [Gelfond, 1960, Lemma II page 135]) Let f be a polynomial in $\mathbb{Z}[x_1, x_2, \ldots, x_n]$ and let d_i be the degree of f in x_i. If g is any factor of f over $\mathbb{Z}$ then $\|g\| \leq \mathrm{e}^{d_1+d_2+\cdots+d_n} \|f\|$ where* $\mathrm{e} = 2.71828$.

Let A be an $m \times m$ matrix with entries $A_{i,j} \in \mathbb{Z}$. The Hadamard bound

$$H(A) = \prod_{i=1}^{m} \sqrt{\sum_{j=1}^{m} A_{i,j}^2}$$

satisfies $|\det A| \leq H(A)$.

Lemma 2.3. *(Goldstein and Graham [Goldstein and Graham, 1974]) Let A be an $m \times m$ matrix with entries $A_{i,j} \in \mathbb{Z}[y]$. Let B be an $m \times m$ integer matrix with entries $B_{i,j} = \|A_{i,j}\|_1$. Then $\|\det A\| \leq H(B)$.*

For polynomials $A = \sum_{i=0}^{s} a_i x_0^i$ and $B = \sum_{i=0}^{t} b_i x_0^i$ where a_i and b_i are in a commutative ring, Sylvester's matrix is the following $s + t$ by $s + t$ matrix

$$S = \begin{bmatrix} a_s & 0 & & 0 & b_t & 0 & & 0 \\ a_{s-1} & a_s & & 0 & b_{t-1} & b_t & & 0 \\ \vdots & a_{s-1} & \ddots & 0 & \vdots & b_{t-1} & \ddots & 0 \\ a_1 & \vdots & & a_s & b_1 & \vdots & & b_t \\ a_0 & a_1 & & a_{s-1} & b_0 & b_1 & & b_{t-1} \\ 0 & a_0 & & \vdots & 0 & b_0 & & \vdots \\ 0 & 0 & \ddots & a_1 & 0 & 0 & \ddots & b_1 \\ 0 & 0 & & a_0 & 0 & 0 & & b_0 \end{bmatrix} \tag{2.1}$$

where the coefficients of A are repeated in the first t columns and the coefficients of B are repeated in the last s columns. The Sylvester resultant of the polynomials A and B in x,

denoted by $\mathrm{res}_x(A, B)$, is the determinant of the Sylvester's matrix. We gather the following facts about the Sylvester resultant into Lemma 2.4 below.

Lemma 2.4. *Let D be any integral domain and let A, B be two polynomials in $D[x_0, \ldots, x_n]$ with $s = \deg_{x_0} A > 0$ and $t = \deg_{x_0} B > 0$. Let $a_s = LC(A)$, $b_t = LC(B)$, $R = \mathrm{res}_{x_0}(A, B)$, $\alpha \in D^n$ and p be a prime. Then*

(i) R is a polynomial in $D[x_1, \ldots, x_n]$,
(ii) $\deg R \leq \deg A \deg B$ (Bezout bound) and
(iii) $\deg_{x_i} R \leq t \deg_{x_i} A + s \deg_{x_i} B$ for $1 \leq i \leq n$.

If D is a field and $a_s(\alpha) \neq 0$ and $b_t(\alpha) \neq 0$ then
(iv) $\mathrm{res}_{x_0}(A(x_0, \alpha), B(x_0, \alpha)) = R(\alpha)$ and
(v) $\deg_{x_0} \gcd(A(x_0, \alpha), B(x_0, \alpha)) > 0 \iff \mathrm{res}_{x_0}(A(x_0, \alpha), B(x_0, \alpha)) = 0$.

If $D = \mathbb{Z}$ and $\phi_p(a_s) \neq 0$ and $\phi_p(b_t) \neq 0$ then
(vi) $\mathrm{res}_{x_0}(\phi_p(A), \phi_p(B)) = \phi_p(R)$ and
(vii) $\deg_{x_0} \gcd(\phi_p(A), \phi_p(B)) > 0 \iff \mathrm{res}_{x_0}(\phi_p(A), \phi_p(B)) = 0$.

Proofs of $(i), (ii), (iv)$ and (v) may be found in Chapter 3 and Chapter 6 of [Cox et al., 1991]. In particular the proof in Chapter 6 of [Cox et al., 1991] for (ii) for bivariate polynomials generalizes to the multivariate case. Note that the condition on α that the leading coefficients a_s and b_t do not vanish implies that the dimension of Sylvester's matrix for $A(x_0, \alpha)$ and $B(x_0, \alpha)$ is the same as that for A and B which proves (v). The same argument used to prove (iv) and (v) works for (vi) and (vii). To prove (iii) we have

$$\deg_{x_i} \det S \leq \sum_{c \in columns(S)} \max_{f \in c} \deg_{x_i} f = \sum_{j=1}^{t} \deg_{x_i} A + \sum_{j=1}^{s} \deg_{x_i} B.$$

2.2 The Ben-Or/Tiwari interpolation

Let $f(x_1, \ldots, x_n) = \sum_{i=1}^{t} a_i M_i \in \mathbb{Z}[x_1, \ldots, x_n]$ be the target polynomial, where $M_i = x_1^{e_{i,1}} \cdots x_n^{e_{i,n}}$, $e_{i,j}$ is the exponent of x_j in M_i and a_i is the nonzero integer coefficient of M_i. Let $t = \#f$ and $m_i = M_i(2, 3, 5, \ldots, p_n)$, where p_n is the nth prime. Let

$$v_j = f(2^j, 3^j, 5^j, \ldots, p_n^j) = a_1 m_1^j + \cdots + a_t m_t^j, \quad \text{for } j = 0, 1, 2, \ldots$$

It is clear that $m_i \neq m_j$ if $i \neq j$. The following observation, which is based on BCH decoding, comprises the core part of the Ben-Or/Tiwari algorithm. We first define the monic polynomial Λ as

$$\Lambda(z) = \prod_{i=1}^{t} (z - m_i) = z^t + \lambda_{t-1} z^{t-1} + \cdots + \lambda_1 z + \lambda_0, \quad \text{where } \lambda_n \in \mathbb{Z} \tag{2.2}$$

We consider the sum

$$\begin{aligned}\sum_{i=1}^{t} a_i m_i^k \Lambda(m_i) &= \sum_{i=1}^{t} a_i m_i^k (m_i^t + \lambda_{t-1} m_i^{t-1} + \cdots + \lambda_1 m_i + \lambda_0) \\ &= \sum_{i=1}^{t} a_i (m_i^{t+k} + \lambda_{t-1} m_i^{t-1+k} + \cdots + \lambda_1 m_i^{k+1} + \lambda_0 m_i^k) \\ &= (\sum_{n=0}^{t-1} \lambda_n (a_1 m_1^{n+k} + a_2 m_2^{n+k} + \cdots + a_t m_t^{n+k})) \\ &\quad + (a_1 m_1^{t+k} + a_2 m_2^{t+k} + \cdots + a_t m_t^{t+k}) \\ &= (\sum_{n=0}^{t-1} \lambda_n v_{n+k}) + v_{t+k},\end{aligned}$$

for $k \geq 0$ and $k \in \mathbb{Z}$. Since m_i are roots of $\Lambda(z)$, we have

$$\sum_{n=0}^{t-1} \lambda_n v_{n+k} + v_{t+k} = \lambda_0 v_k + \lambda_1 v_{k+1} + \cdots + \lambda_{t-1} v_{k+t-1} + v_{t+k} = 0. \tag{2.3}$$

There are t unknowns $\lambda_0, \ldots, \lambda_{t-1}$. Therefore t equations are required to determine the solution. The set of λ_i in Equation (2.3) behaves as a *linear shifter* in BCH decoding. If k runs from 0 to $t-1$, we obtain the linear system

$$\begin{bmatrix} v_0 & v_1 & v_2 & \ldots & v_{t-1} \\ v_1 & v_2 & v_3 & \ldots & v_t \\ \vdots & & \cdots & & \vdots \\ v_{t-2} & v_{t-1} & v_t & \ldots & v_{2t-3} \\ v_{t-1} & v_t & v_{t+1} & \ldots & v_{2t-2} \end{bmatrix} \begin{bmatrix} \lambda_0 \\ \lambda_1 \\ \vdots \\ \lambda_{t-2} \\ \lambda_{t-1} \end{bmatrix} = \begin{bmatrix} -v_t \\ -v_{t+1} \\ \vdots \\ -v_{2t-2} \\ -v_{2t-1} \end{bmatrix}.$$

The $r \times r$ matrix

$$\begin{bmatrix} v_0 & v_1 & v_2 & \ldots & v_{r-1} \\ v_1 & v_2 & v_3 & \ldots & v_r \\ \vdots & & \cdots & & \vdots \\ v_{r-2} & v_{r-1} & v_r & \ldots & v_{2r-3} \\ v_{r-1} & v_r & v_{r+1} & \ldots & v_{2r-2} \end{bmatrix}$$

is called the *square Hankel matrix* of size r and denoted by H_r. If $r = t$ and H_t is invertible, then the linear system has a unique solution. But in most applications t is an unknown. Kaltofen [E Kaltofen and Lobo, 2000] showed that $\det H_r$ is *generically* nonzero when $r \leq t$ and zero when $r > t$. Once variables are evaluated at integers, this property gives us a method to determine t as follows:

1. If $r = t$, then H_r is invertible with rank t, so the linear system has an unique solution;

2. If $r > t$, then H_r is singular and the rank is t;

3. If $r < t$, then H_r is invertible if $\det H_r(2, 3, \ldots, p_n) \neq 0$. See Section 2.5 for more detail.

Gaussian elimination costs $O(t^3)$ arithmetic operations to invert H_t. But the linear shifter can be solved by the Berlekamp-Massey algorithm with $O(t^2)$ arithmetic operations. Therefore the practical version of the Ben-Or/Tiwari interpolation usually applies the Berlekamp-Massey algorithm [Massey, 1969] to determine $\Lambda(z)$.

Once $\Lambda(z)$ is determined, we factor Λ to obtain roots $m_0, \ldots, m_{t-1}$. For each m_i, we use repeated trial division by $p_1 = 2, p_2 = 3, \ldots, p_n$ to get the correct exponent for each variable in M_i. For example, assume $n = 3$. If $M_i = 45000 = 2^3 3^2 5^4$, then $M_i = x_1^3 x_2^2 x_3^4$.

The final step is to determine the coefficients c_i. Since m_i and v_j are known now, we simply construct a linear system based on equations

$$\{a_1 m_1^j + \cdots + a_t m_t^j = v_j, \quad \text{for } 0 \leq j \leq t-1.\} \tag{2.4}$$

Or explicitly we have

$$\begin{bmatrix} 1 & 1 & 1 & \ldots & 1 \\ m_1 & m_2 & m_3 & \ldots & m_t \\ m_1^2 & m_2^2 & m_3^2 & \ldots & m_t^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m_1^{t-1} & m_2^{t-1} & m_3^{t-1} & \ldots & m_t^{t-1} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_t \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{t-1} \end{bmatrix}$$

The matrix above is a transposed Vandermonde matrix and we name it V. Recall that

$$\det V = \det V^T = \prod_{1 \leq j < k \leq t} (m_j - m_k).$$

Since $m_i = M_i(2, 3, 5, \ldots, p_n) \neq M_j(2, 3, 5, \ldots, p_n) = m_j$ for $i \neq j$, it follows that the linear system (2.4) has a unique solution and can be solved by $O(t^3)$ arithmetic operations naively. However Zippel's Vandermonde system solving algorithm can reduce the cost to $O(t^2)$ arithmetic operations and $O(t)$ space. We will discuss this in Section 2.7. The Ben-Or/Tiwari sparse interpolation algorithm is presented in Figure 2.1

The Ben-Or/Tiwari algorithm has the intermediate expression swell problem. For example, the constant term $\prod_{i=1}^{t} m_i$ of $\Lambda(z)$ is of size $tn \log(\deg f)$ bits. It is worth mentioning that $\Lambda(z)$ can also be computed by Sugiyama algorithm [Sugiyama et al., 1975] (the Euclidean algorithm). However a severe expression swell may occur when we run either the Berlekamp-Massey algorithm [Massey, 1969] or the Sugiyama algorithm to compute $\Lambda(z)$ over $\mathbb{Q}$. For our purposes, since we want to design a modular algorithm, steps 2, 3 and 5 run modulo a prime p. If $p > p_n^D \geq \max_{i=1}^{t} m_i$, the integers $m_i \bmod p$ remain unique. For step

Ben-Or/Tiwari interpolation algorithm
Inputs: A function $f(x_1, \ldots, x_n)$ which computes the polynomial
$f(x_1, \ldots, x_n) = \sum_{i=1}^{t} a_i M_i$ where $a_i \in \mathbb{Z}$ and M_i are monomials in $\mathbb{Z}[x_1, \ldots, x_n]$.
Output: The terms $a_i M_i$ for $1 \leq i \leq t$.

1. Compute $v_i = f(2^i, 3^i, \ldots, p_n^i)$ for $0 \leq i \leq 2t-1$.
2. Compute $\Lambda(z)$ from v_i by using the Berlekamp-Massey algorithm.
3. Compute the roots $m_1, \ldots, m_t$ of $\Lambda(z)$.
4. Factor each root m_i by trial division by $2, 3, \ldots, p_n$, recover $e_{i,j}$ where $1 \leq j \leq n$.
5. Solve the linear system $\{a_1 m_1^j + \cdots + a_t m_t^j = v_j\}_{0 \leq j \leq t-1}$ and determine the unknown coefficients a_i.
6. Output $a_i M_i$ for $1 \leq i \leq t$.

Figure 2.1: The Ben-Or/Tiwari interpolation algorithm

4, the roots of $\Lambda(z) \in \mathbb{Z}_p[z]$ can be found by Berlekamp's algorithm which has the classical complexity $O(t^2 \log p)$. See [Berlekamp, 1970, Rabin, 1979].

The Ben-Or/Tiwari algorithm assumes that a sparse term bound $\tau \geq t$ is known. Therefore in step 1 we may compute 2τ evaluations in parallel. In practice such a bound on t may not be given in advance so the algorithm needs to be modified to also determine t. For p sufficiently large, if we compute $\Lambda(z)$ after $j = 2, 4, 6, \ldots$ points, we will see $\deg \Lambda(z) = 1, 2, 3, \ldots, t-1, t, t, t, \ldots$ with high probability. Thus we may simply wait until the degree of $\Lambda(z)$ does not change. This approach was first discussed by Kaltofen, Lee and Lobo in [E Kaltofen and Lobo, 2000]. We will return to this in Section 2.5.

Steps 2, 3, and 5 may be accelerated with fast multiplication. Let $M(t)$ denote the cost of multiplying two polynomials of degree t in $\mathbb{Z}_p[t]$. The fast Euclidean algorithm can be used to accelerate step 2 if we use the Sugiyama algorithm which applies the Euclidean algorithm to determine $\Lambda(z)$. Therefore it has complexity $O(M(t) \log t)$. See [von zur Gathen and Gerhard, 1999, Chapter 11]. Computing the roots of $\Lambda(z)$ in step 3 can be done in $O(M(t) \log t \log pt)$. See [von zur Gathen and Gerhard, 1999, Chapter 14]. Step 5 can be done in $O(M(t) \log t)$ using fast interpolation. See [von zur Gathen and Gerhard, 1999, Chapter 10]. We summarize these complexity results in Table 2.2 below.

Step	Classical	Fast
2	$O(t^2)$	$O(M(t) \log t)$
3	$O(t^2 \log p)$	$O(M(t) \log t \log pt)$
5	$O(t^2)$	$O(M(t) \log t)$

Table 2.2: Number of arithmetic operations in $\mathbb{Z}_p$ for t monomials.

2.3 Ben-Or/Tiwari algorithm with discrete logarithm

We first introduce the discrete logarithm method we propose to use. Let $f(x_1, \ldots, x_n) = \sum_{i=1}^{t} a_i M_i \in \mathbb{Z}[x_1, \ldots, x_n]$ and $d_i = \deg_{x_i} f$. We choose a prime p which has the form $p = q_1 \times q_2 \times \cdots q_n + 1$ where $q_i > d_i$ and $\gcd(q_i, q_j) = 1$ for $i \neq j$. If $q_i \leq y$ are small then p is called *y-smooth*. We randomly pick a generator ω of $\mathbb{Z}_p^*$ and set $w_j = \omega^{\frac{p-1}{q_j}}$ for $1 \leq j \leq n$. Therefore all w_js have relatively prime orders. We run the Berlekamp-Massey algorithm on the sequence $[f(w_1^k, \ldots, w_n^k) : 0 \leq k \leq 2t-1]$ to get a feedback polynomial $c(z)$. Let $\Lambda(z)$ be the reciprocal of $c(z)$. Next we factor $\Lambda(z)$ to obtain roots v_k for $1 \leq k \leq t$. The exponent of each variable in each monomial is recovered as follows:

$$\begin{aligned}
m_k &= M_k(w_1, w_2, \ldots, w_n) = w_1^{e_{k1}} \times w_2^{e_{k2}} \times \cdots \times w_n^{e_{kn}} \\
&= \omega^{\frac{p-1}{q_1} e_{k1}} \times \omega^{\frac{p-1}{q_2} e_{k2}} \times \cdots \times \omega^{\frac{p-1}{q_n} e_{kn}} = \omega^{\frac{p-1}{q_1} e_{k1} + \cdots + \frac{p-1}{q_n} e_{kn}}, \\
&\Rightarrow \log_\omega m_k = \frac{p-1}{q_1} e_{k1} + \cdots + \frac{p-1}{q_i} e_{ki} + \cdots + \frac{p-1}{q_n} e_{kn} \mod p-1, \qquad (2.5) \\
&\Rightarrow \log_\omega m_k = 0 + \cdots + \frac{p-1}{q_i} e_{ki} + \cdots + 0 \mod q_i, \\
&\Rightarrow e_{ki} = \log_\omega m_k \times \frac{q_i}{p-1} \mod q_i.
\end{aligned}$$

Note that the condition $\gcd(q_i, q_j) = 1$ ensures that $\frac{p-1}{q_k}$ is invertible modulo q_k. The original Ben-Or/Tiwari algorithm evaluates inputs at primes, $m_i = M_i(2, 3, \ldots, p_n) \neq M_j(2, 3, \ldots, p_n) = m_j$ for $i \neq j$ because m_i and m_j have different prime factorizations. But it is not obvious that $i \neq j$ implies $m_i \neq m_j$ if we choose to evaluate at $w_1, \ldots, w_n$. The following proposition guarantees that this key feature is preserved in the discrete logarithm method.

Proposition 2.1. *In the discrete logarithm method, let $q_i > d_i$. If $i \neq j$ then*

$$m_i = M_i(w_1, \ldots, w_n) \neq M_j(w_1, \ldots, w_n) = m_j.$$

Proof. First we recall that $i \neq j$ implies $M_i \neq M_j$. We work toward a contradiction by assuming that $m_i = M_i(w_1, \ldots, w_n) = M_j(w_1, \ldots, w_n) = m_j$ and $i \neq j$. Then we have

$$\begin{aligned}
\omega^{\frac{e_{i,1}(p-1)}{q_1}} \cdots \omega^{\frac{e_{i,n}(p-1)}{q_n}} &\equiv \omega^{\frac{e_{j,1}(p-1)}{q_1}} \cdots \omega^{\frac{e_{j,n}(p-1)}{q_n}} \mod p \\
&\Updownarrow \\
\omega^{\frac{e_{i,1}(p-1)}{q_1} + \cdots + \frac{e_{i,n}(p-1)}{q_n}} &\equiv \omega^{\frac{e_{j,1}(p-1)}{q_1} + \cdots + \frac{e_{j,n}(p-1)}{q_n}} \mod p \\
&\Updownarrow \\
\frac{e_{i,1}(p-1)}{q_1} + \cdots + \frac{e_{i,n}(p-1)}{q_n} &\equiv \frac{e_{j,1}(p-1)}{q_1} + \cdots + \frac{e_{j,n}(p-1)}{q_n} \mod p-1
\end{aligned}$$

The above equation is equivalent to

$$\frac{e_{i,1}(p-1)}{q_1} + \cdots + \frac{e_{i,n}(p-1)}{q_n} = \frac{e_{j,1}(p-1)}{q_1} + \cdots + \frac{e_{j,n}(p-1)}{q_n} + (p-1)t$$

for some $t \in \mathbb{Z}$. Since $\gcd(q_i, q_j) = 1$ for $1 \leq i \neq j \leq n$ and $p - 1 = q_1 \cdots q_n$, the above equation becomes

$$\frac{e_{i,k}(p-1)}{q_k} \equiv \frac{e_{j,k}(p-1)}{q_k} \mod q_k$$

for $1 \leq k \leq n$. Since $\gcd(\frac{p-1}{q_k}, q_k) = 1$, by the cancellation law we have

$$e_{i,k} \equiv e_{j,k} \mod q_k.$$

Since $e_{i,k} \leq d_k < q_k$ and $e_{j,k} \leq d_k < q_k$, we have $e_{i,k} = e_{j,k}$ for $1 \leq k \leq n$. Therefore $M_i = M_j$, which contradicts our assumption $M_i \neq M_j$.

□

The discrete logarithm method modifies the Ben-Or/Tiwari interpolation algorithm so that the prime needed is a little larger than $(d+1)^n$ bits. As a result the size of the prime is $O(n \log d)$ bits instead of $O(D \log n \log \log n)$ bits where $D = \deg f$.

Example 2.2. *Let $f(x_1, \ldots, x_{10})$ be a multivariate polynomial in 10 variables and each variable has degree at most 5. In order to satisfy the condition that $m_i \neq m_j$ for $i \neq j$, the Ben-Or/Tiwari's algorithm must pick a prime p so that*

$$\begin{aligned} p &\geq (2^6 \times 3^6 \times 5^6 \times \cdots \times 29^6) + 1 \\ &= 733334523056958157244045495461478753876403151410052890000001. \end{aligned}$$

For the discrete logarithm method with $q_i > d_i$, the prime $p \geq 6^{10} + 1$. A prime satisfying the condition for the discrete logarithms method is

$$p = 6 \cdot 7 \cdot 11 \cdot 13 \cdot 17 \cdot 19 \cdot 23 \cdot 25 \cdot 29 \cdot 31 + 1 = 1002802450651.$$

In the discrete logarithm method a prime should be $y-$smooth. For example, $y = 100$ or $y = 1000$ should be fine. Then the Pohlig-Hellman algorithm [Pohlig and Hellman, 1978] is able to compute the discrete logarithm efficiently. Suppose $i \neq j$. To get such a smooth prime, we first compute a list of integers $s = [s_1, s_2, \ldots, s_n]$ so that $s_i > \deg_{x_i} f$ and $\gcd(s_i, s_j) = 1$, then apply the prime number test to $k \times s_1 s_2 \cdots s_n + 1$ for $k = 1, 2, 3, \ldots$. Suppose $K \times s_1 s_2 \cdots s_n + 1$ is a prime, then we loop i from 1 to n by updating s_i with $s_i \gcd(K, s_i)$ and setting $K = K/\gcd(K, s_i)$. At last, if $K \neq 1$, we set $s_j = K s_j$ where s_j is the smallest number in the updated list s. By setting $q_i = s_i$ we obtain a prime with the

form $p = q_1 q_2 \cdots q_n + 1$ and $\gcd(q_i, q_j) = 1$. The next example illustrates the process to get a smooth prime.

Example 2.3. *Suppose* $s = [12, 13, 17, 19, 23]$. *We set* $p = K \times 12 \cdot 13 \cdot 17 \cdot 19 \cdot 23 + 1$. *We first find out* p *is a prime when* $K = 20$. *Since* $\gcd(20, s_1) = 4$, *we update* $s_1 = 4 \cdot 12 = 48$ *and set* $K = K/4 = 5$. *Since* 5 *is coprime with* s_i *for* $2 \leq i \leq 5$ *and* $s_2 = 13$ *is the smallest in the updated* s, *we set* $s_2 = 13 \cdot 5 = 65$. *Therefore the smooth prime is* $p = 48 \cdot 65 \cdot 17 \cdot 19 \cdot 23 + 1$.

Theorem 2.1 (Dirichlet,1837)**.** *If* a *and* d *are coprime integers and* $d > 0$, *then the arithmetic progression* $\{a, a + d, a + 2d, \dots\}$ *contains infinitely many primes.*

If we set $a = s_1 s_2 \cdots s_n + 1$ and $d = s_1 s_2 \cdots s_n$, then there are infinitely many primes having the form $k \times s_1 s_2 \cdots s_n + 1$. Suppose p is a prime and the prime factorization of $p - 1$ is

$$p - 1 = p_1^{c_1} p_2^{c_2} \cdots p_k^{c_k}.$$

The total cost to solve a discrete logarithm by the Pohlig-Hellman algorithm is

$$O(\sum_{i=1}^{k} c_i(\log_2 p + \sqrt{p_i})) + O(k \log_2 p) \in O(\sum_{i=1}^{k} c_i(\log_2 p + \sqrt{p_i}))$$

where $O(k \log_2 p)$ is contributed by the Chinese remainder theorem. See [Pohlig and Hellman, 1978, Section 4]. In our problem, the prime has the form $p - 1 = q_1 q_2 \cdots q_n$ and $\gcd(q_i, q_j) = 1$ if $i \neq j$. We decompose $q_k = q_{k,1}^{e_{k,1}} q_{k,2}^{e_{k,2}} \cdots q_{k,m}^{e_{k,m}}$ hence $m \leq \lceil \log_2 q_k \rceil$. To compute $\log_\omega \beta \mod q_k$, Pohlig-Hellman costs $O(\sum_{i=1}^{m} e_{k,i}(\log_2 p + \sqrt{q_{k,i}}))$. We note that $q_{k,i} \leq q_k$. For some k if $q_{k,i} \approx q_k$, then $e_{k,i}$ and m are negligible hence $O(\sum_{i=1}^{m} e_{k,i}(\log_2 p + \sqrt{q_{k,i}})) \in O(\log_2 p + \sqrt{q_k})$. If $q_{k,j} \ll q_k$ (q_k can not be too large because p is assumed to be y-smooth hence $q_{k,j}$ is small), then $\sqrt{q_{k,i}} \approx \log_2 q_{k,i}$ hence $O(\sum_{i=1}^{m} e_{k,i}(\log_2 p + \sqrt{q_{k,i}})) \approx O(\sum_{i=1}^{m} e_{k,i} \log_2 p + \sum_{i=1}^{m} e_{k,i} \log_2 q_{k,i}) = O(\sum_{i=1}^{m} e_{k,i} \log_2 p + \log_2 q_k) \in O(\sum_{i=1}^{m} e_{k,i} \log_2 p + \sqrt{q_k})$. Since $\sum_{j=1}^{n} \sum_{i=1}^{m} e_{j,i} \leq \log_2 p$ which is the upper bound for the number of prime factors of $p - 1$, by combining both cases above, the total cost to compute $\log_\omega \beta \mod p$ is

$$O(\sum_{j=1}^{n} \sum_{i=1}^{m} e_{j,i} \log_2 p + \sum_{j=1}^{n} \sqrt{q_j}) \in O((\log_2 p)^2 + \sum_{j=1}^{n} \sqrt{q_j}).$$

Let $d \geq \max_{i=1}^{n}(\deg_{x_i} f)$. Kaltofen showed in [Kaltofen, 2010] that computing discrete logarithm can be made polynomial in $\log d$ and n if one uses a Kronecker substitution to reduce a multivariate interpolation to a univariate interpolation and uses a prime $p > (d+1)^n$ of the form $p = 2^k s + 1$ with s small.

Once $\Lambda(z)$ is determined by the Berlekamp-Massey algorithm, we need to compute its roots $m_1, m_2, \dots m_t$. This can be done efficiently by the Berlekamp's root finding algorithm [Berlekamp, 1970, Rabin, 1979]. It has the classical complexity $O(t^2 \log p)$ which can be

reduced to $O(M(t)\log pt\log t)$ if fast multiplication is applied where $M(t)$ is the cost of multiplication. See [von zur Gathen and Gerhard, 1999, Chapter 14].

At the end of this section, we discuss the complexity of the modular Ben-Or/Tiwari interpolation algorithm with discrete logarithm method. We count the number of arithmetic operations in $\mathbb{Z}_p$. Let $f(x_1,\ldots,x_n)\in\mathbb{Z}[x_1,\ldots,x_n]$ be the target polynomial to be interpolated. Let $d_i=\deg_{x_i} f$ and $d=\max_{i=1}^{m}\{d_i\}$. A prime p of size $p>(d+1)^n$ is large enough to recover all exponents of monomials appearing in f. We assume the term bound $\tau\geq t=\#f$ and a generator ω of $\mathbb{Z}_p^*$ are given. See [Geddes et al., 1992, Theorem 4.5] for how to compute a generator. Let $P(n,d,t)$ be the cost of one evaluation of $f(\beta_1,\ldots,\beta_n)$ mod p. The computation of ω^j for $0\leq j<2\tau$ takes $O(2\tau n)$ operations. Hence the total cost to evaluate f 2τ times is $O(\tau n+\tau P(n,d,t))$. The Berlekamp-Massey algorithm on an input sequence of length 2τ has complexity $O(\tau^2)$. See [van Tilborg and Jajodia, 2011, Page 80]. The Berlekamp-Massey algorithm also determines t. Therefore we use t instead of τ now. The root finding can be done in $O(t^2\log p)$. The discrete logarithm solving costs $O((\log_2 p)^2+\sum_{i=1}^{n}\sqrt{q_i})$. To recover exponents, we compute

$$e_{ki}=\log_\omega v_k\times\frac{q_i}{p-1}\mod q_i.$$

There are one inversion for $(p-1)^{-1}$ and n multiplications for $q_i(p-1)^{-1}$ where $1\leq i\leq n$. Since there are t monomials to recover, the total cost to compute exponents is $O(t(n+(\log_2 p)^2+\sum_{i=1}^{n}\sqrt{q_i}))$. Zippel solves the Vandermonde linear system in $O(t^2)$. Therefore we have the following theorem.

Theorem 2.2. *The expected number of arithmetic operations in the modular Ben-Or/Tiwari interpolation with discrete logarithms method over $\mathbb{Z}_p$ is*

$$O(\tau n+\tau P(n,d,t)+\tau^2+t^2\log p+t(n+(\log_2 p)^2+\sum_{i=1}^{n}\sqrt{q_i})).$$

2.4 A first example

We have reviewed the modular Ben-Or/Tiwari interpolation algorithm with the discrete logarithm method. Now we need to apply it to the polynomial GCD problem. We use the following example to demonstrate the design of our new GCD algorithm. Recall that $H=\sum_{i=0}^{d_G}h_i x_0^i=\Delta G$ where $\deg_{x_0} G=d_G$. In our algorithm there are two ways to obtain the modular image $H \mod p$. The first method using the Ben-Or/Tiwari interpolation with the discrete logarithm method is the core of our new algorithm. In this method primes are smooth and the size of primes depends on degrees of variables. $t=\max_{i=0}^{d_G}\{\#h_i\}$ is an unknown and needs to be determined. Several algorithms are called to serve the intermediate steps. Hence a lot of things can go wrong. The main purpose of the first method is to determine the support of the target GCD. The second method is the second part of Zippel's

sparse interpolation. That is to solve linear systems to obtain more images, provided that the support obtained by the first method is correct. Then we apply the Chinese remainder theorem to all modular images obtained by the first and the second methods to recover the integer coefficients. In our implementation with 63 bits or 127 bits primes, one run of the first method would determine all monomials in $Supp(H)$ with high probability.

We mention again that we require inputs A and B to be primitive with respect to the main variable we choose. In this thesis, x or x_0 are assumed to be the main variable. If A or B is not primitive in x then we could use our GCD algorithm to compute the content of all coefficients of A or B and divide out the content to pre-process inputs A and B.

Example 2.4. *Consider the GCD problem $A = \bar{A}G$ and $B = \bar{B}G$, where*

$$G = (92y^2 - 513z)x^2 + (212y^2 + yz^2 + 125z)x + (251y^2z^2 - 43z^3 + 5y^2 + 318),$$

$$\bar{A} = y^2x + z \quad \textit{and} \quad \bar{B} = y^3x^2 + z.$$

Let x be the main variable and $\Gamma = \gcd(LC(A), LC(B)) = y^2(92y^2 - 513z)$. It is easy to check that A and B are primitive with respect to x. In order to pick a smooth prime, we first compute the degree bounds of y and z in the target GCD. We randomly pick a prime $p = 53$ and choose 3 evaluation points $28, 46, 44$ uniformly at random from $\mathbb{Z}_{53}$ for x, y, z, respectively. We compute

$$\gcd(A(x = 28, y, z = 44), B(x = 28, y, z = 44)) \mod 53 = y^2 + 51y + 18,$$

$$\gcd(A(x = 28, y = 46, z), B(x = 28, y = 46, z)) \mod 53 = z^3 + 39z^2 + 8z.$$

Since $LC(A)(x = 28, y, z = 44)$ and $LC(A)(x = 28, y = 46, z)$ do not vanish modulo p, $\deg_y G \leq 2$ and $\deg_z G \leq 3$. Since $\deg_y \Gamma G = \deg_y \Gamma + \deg_y G$ and $\deg_z \Gamma G = \deg_z \Gamma + \deg_z G$, the smooth prime should have the form $p = q_1q_2 + 1$ where $q_1 > 2 + \deg_y \Gamma = 6$ and $q_2 > 3 + \deg_z \Gamma = 4$. It is not hard to find out that we can use $q_1 = 14$ and $q_2 = 15$ so that $p = 14 \times 15 + 1 = 211$ is a prime and $\gcd(14, 15) = 1$. We randomly pick the generator $\omega = 2$ in $\mathbb{Z}_p^*$ and set up the evaluation points for y and z as

$$\omega_1 = \omega^{\frac{p-1}{q_1}} \equiv 63 \mod p \quad \textit{and} \quad \omega_2 = \omega^{\frac{p-1}{q_2}} \equiv 137 \mod p.$$

We compute the first 2 scaled GCD images and get

$$g_0 = \Gamma(\omega_1^0, \omega_2^0) \cdot \gcd(A(x, 63^0, 137^0), B(x, 63^0, 137^0)) = x^2 + 127x + 109.$$
$$g_1 = \Gamma(\omega_1^1, \omega_2^1) \cdot \gcd(A(x, 63^1, 137^1), B(x, 63^1, 137^1)) = 9x^2 + 120x + 144.$$

We extract the coefficients of x in g_0, g_1 to form a sequence $[127, 120]$ *and do the same for the constant term and get* $[109, 144]$. *We run the Berlekamp-Massey algorithm (BMA) on* $[127, 120]$ *and* $[109, 144]$ *and obtain* $c_1(z) = 122z + 1$ *and* $c_2(z) = 200z + 1$ *respectively. We compute the next two scaled GCD images and get*

$$g_2 = \Gamma(\omega_1^2, \omega_2^2) \cdot \gcd(A(x, 63^2, 137^2), B(x, 63^2, 137^2)) = x^2 + 58x + 194.$$
$$g_3 = \Gamma(\omega_1^3, \omega_2^3) \cdot \gcd(A(x, 63^3, 137^3), B(x, 63^3, 137^3)) = 28x^2 + 140x + 20.$$

We extract the coefficient of x in g_1, g_2, g_2, g_3 to form the sequence $[127, 120, 58, 140]$ *and the coefficient of the constant term to form the sequence* $[109, 104, 194, 20]$. *We run BMA on both sequences and get* $c_1(z) = 102z^2 + 194z + 1$ *and* $c_2(z) = 83z^2 + 34z + 1$. *The degrees of both feedback polynomials increased by* 1 *hence neither* c_i *has stabilized and we compute the next two scaled GCD images*

$$g_4 = \Gamma(\omega_1^4, \omega_2^4) \cdot \gcd(A(x, 63^4, 137^4), B(x, 63^4, 137^4)) = 131x^2 + 84x + 59$$
$$g_5 = \Gamma(\omega_1^5, \omega_2^5) \cdot \gcd(A(x, 63^5, 137^5), B(x, 63^5, 137^5)) = 33x^2 + 105x + 201.$$

The coefficient sequences are updated to

$$[127, 120, 58, 140, 84, 105] \quad \textit{and} \quad [109, 104, 194, 20, 59, 201].$$

The feedback polynomials returned by the BMA are $c_1(z) = 151z^3 + 28z^2 + 202z + 1$ *and* $c_2(z) = 195z^3 + 187z^2 + 18z + 1$ *which have not stabilized hence we compute the next two scaled GCD images*

$$g_6 = \Gamma(\omega_1^6, \omega_2^6) \cdot \gcd(A(x, 63^6, 137^6), B(x, 63^6, 137^6)) = 208x^2 + 30x + 186$$
$$g_7 = \Gamma(\omega_1^7, \omega_2^7) \cdot \gcd(A(x, 63^7, 137^7), B(x, 63^7, 137^7)) = 157x^2 + 49x + 153.$$

The coefficient sequences are updated to

$$[127, 120, 58, 140, 84, 105, 30, 49] \quad \textit{and} \quad [109, 104, 194, 20, 59, 201, 186, 153].$$

The feedback polynomials returned by the BMA are $c_1(z) = 151z^3 + 28z^2 + 202z + 1$ *and* $c_2(z) = 185z^4 + 10z^3 + z^2 + 152z + 11$. $c_1(z)$ *did not change. But* $c_2(z)$ *is still not stable, we compute two more scaled GCD images*

$$g_8 = \Gamma(\omega_1^8, \omega_2^8) \cdot \gcd(A(x, 63^8, 137^8), B(x, 63^8, 137^8)) = 101x^2 + 204x + 136$$
$$g_9 = \Gamma(\omega_1^9, \omega_2^9) \cdot \gcd(A(x, 63^9, 137^9), B(x, 63^9, 137^9)) = 131x^2 + 154x + 54.$$

The sequence corresponding to the constant term is updated to

$$[109, 104, 194, 20, 59, 201, 186, 153, 136, 54]$$

on which we run the BMA and get $c_2(z) = 185z^4 + 10z^3 + z^2 + 152z + 11$. $c_2(z)$ *has now stabilized and we terminate the looping. It is likely that there are* 3 *non-zero terms in the coefficient of* x *and* 4 *non-zero terms in the constant term. We compute the reciprocals (the reason is given in the next section) of both stable feedback polynomials which are* $28z^3 + 54z^2 + 38z + 1$ *and* $8z^4 + 53z^3 + 6z^2 + 59z + 1$ *to get*

$$\Lambda_1 = z^3 + 202z^2 + 28z + 151 \quad \textit{and} \quad \Lambda_2 = z^4 + 152z^3 + z^2 + 10z + 185.$$

The roots of Λ_1 *are* $[6, 91, 123]$ *in* $\mathbb{Z}_p$ *and the roots of* Λ_2 *are* $[151, 36, 171, 123]$ *in* $\mathbb{Z}_p$. *We convert the roots to monomials by the discrete logarithm method. For instance, the conversion of the root* 6 *of* Λ_1 *is*

$$\log_\omega 6 \cdot \frac{q_1}{p-1} = 44 \cdot \frac{14}{210} = 2 \mod 14 \textit{ and } \log_\omega 6 \cdot \frac{q_2}{p-1} = 44 \cdot \frac{15}{210} = 1 \mod 15.$$

Hence the root 6 *corresponds to the monomial* y^2z. *Similar arguments show*

$$[6, 91, 123] \Longrightarrow [y^2z, y^3z^2, y^4] \quad \textit{and} \quad [151, 36, 171, 123] \Longrightarrow [y^2z^3, y^4z^2, y^2, y^4].$$

The final step is to determine the integer coefficient of each monomial modulo p. *That is to solve the following two Vandermonde linear systems*

$$\begin{bmatrix} 6^0 & 91^0 & 123^0 \\ 6^1 & 91^1 & 123^1 \\ 6^2 & 91^2 & 123^2 \end{bmatrix} \begin{bmatrix} C_{1,1} \\ C_{1,2} \\ C_{1,3} \end{bmatrix} = \begin{bmatrix} 127 \\ 120 \\ 58 \end{bmatrix},$$

$$\begin{bmatrix} 151^0 & 36^0 & 171^0 & 123^0 \\ 151^1 & 36^1 & 171^1 & 123^1 \\ 151^2 & 36^2 & 171^2 & 123^2 \\ 151^3 & 36^3 & 171^3 & 123^3 \end{bmatrix} \begin{bmatrix} C_{2,1} \\ C_{2,2} \\ C_{2,3} \\ C_{2,4} \end{bmatrix} = \begin{bmatrix} 109 \\ 144 \\ 194 \\ 20 \end{bmatrix}.$$

The solutions to the linear systems are $C_{1,1} = 125$, $C_{1,2} = 1$, $C_{1,3} = 1$, $C_{2,1} = 168$, $C_{2,2} = 40$, $C_{2,3} = 107$, $C_{2,4} = 5$. *Hence we have the first GCD image* $H_1 = H \mod p$ *with coefficients in the symmetric representation for* $\mathbb{Z}_p$

$$H_1 = (92y^4 + 120y^2z)x^2 + (y^3z^2 + y^4 + 125y^2z)x + 40y^4z^2 + 168y^2z^3 + 5y^4 + 107y^2.$$

Note that the coefficient of x^2 is the scale factor Γ. Hence no calculation is required. Now let us assume $Supp(H_1) = Supp(H)$ and the assumed form is

$$H_{form} = (T_{1,1}y^4 + T_{1,2}y^2z)x^2 + (T_{2,1}y^3z^2 + T_{2,2}y^4 + T_{2,3}y^2z)x + T_{3,1}y^4z^2 + T_{3,2}y^2z^3 + T_{3,3}y^4 + T_{3,4}y^2.$$

With the assumed form, we use Zippel's sparse interpolation to compute the next multivariate GCD image modulo a new prime. The maximum number of non-zero terms in coefficients with respect to x is 4, in the constant term. Hence we need to choose 4 evaluation points. We pick the prime $p = 101$ at random which does not divide the leading coefficient of A or B and compute $H'(x, \alpha, \beta) = \Gamma(x, \alpha, \beta)\gcd(A(x, \alpha, \beta), B(x, \alpha, \beta))$ with 4 uniformly and randomly chosen evaluation points from $\mathbb{Z}_{101}$.

α	β	
16	100	$H'(x, 16, 100) = 44x^2 + 44x + 6$
89	33	$H'(x, 89, 33) = 85x^2 + 66x + 89$
17	51	$H'(x, 17, 51) = 9x^2 + 66x + 90$
55	42	$H'(x, 55, 42) = 41x^2 + 61x + 63$

By equating the coefficients of $H_{form}(x, \alpha, \beta)$ with the coefficients of $H'(x, \alpha, \beta)$, we obtain following linear systems. For the coefficient of x^2 we have

$$25T_{1,1} + 93T_{1,2} = 41, \quad 31T_{1,1} + 5T_{1,2} = 85,$$
$$88T_{1,1} + 47T_{1,2} = 44, \quad 95T_{1,1} + 94T_{1,2} = 9,$$

which has a solution $T_{1,1} = 92$ and $T_{1,2} = 93$. For the coefficient of x^1 we have

$$3T_{2,1} + 25T_{2,2} + 93T_{2,3} = 61, \quad 40T_{2,1} + 31T_{2,2} + 5T_{2,3} = 66,$$
$$56T_{2,1} + 88T_{2,2} + 47T_{2,3} = 44, \quad 92T_{2,1} + 95T_{2,2} + 94T_{2,3} = 66,$$

which has a solution $T_{2,1} = 1, \quad T_{2,2} = 10, \quad T_{2,3} = 24$. For the constant term we have

$$25T_{3,1} + 92T_{3,2} + 31T_{3,3} + 43T_{3,4} = 89, \quad 49T_{3,1} + 74T_{3,2} + 95T_{3,3} + 87T_{3,4} = 90,$$
$$64T_{3,1} + 28T_{3,2} + 25T_{3,3} + 96T_{3,4} = 63, \quad 88T_{3,1} + 47T_{3,2} + 88T_{3,3} + 54T_{3,4} = 6,$$

which has a solution $T_{3,1} = 49, \quad T_{3,2} = 58, \quad T_{3,3} = 5, \quad T_{3,4} = 15$. Hence the second image $H_2 = H \mod 101$ is

$$H_2 = (92y^4 + 93y^2z)x^2 + (y^3z^2 + 10y^4 + 24y^2z)x + 49y^4z^2 + 58y^2z^3 + 5y^4 + 15y^2 \mod 101.$$

We apply the Chinese remaindering to $H_1 \mod 211$ *and* $H_2 \mod 101$ *with* the symmetric coefficient representation *modulo* 211×101 *and obtain*

$$\hat{H}_1 = (92y^4 - 513y^2z)x^2 + (y^3z^2 + 212y^4 + 125y^2z)x + 251y^4z^2 - 43y^2z^3 + 5y^4 + 318y^2.$$

Since $H_1 \neq \hat{H}_1$ *or* $H_2 \neq \hat{H}_1$*, we compute the next image and choose the prime* $p = 103$ *at random. With the identical steps we had to determine* H_2 *modulo* 103*, we get*

$$H_3 = (92y^4 + 2y^2z)x^2 + (y^3z^2 + 6y^4 + 22y^2z)x + 45y^4z^2 + 60y^2z^3 + 5y^4 + 9y^2 \mod 103.$$

We apply the Chinese remaindering to $\hat{H}_1$ *and* H_3 *and get*

$$\hat{H}_2 = (92y^4 - 513y^2z)x^2 + (y^3z^2 + 212y^4 + 125y^2z)x + 251y^4z^2 - 43y^2z^3 + 5y^4 + 318y^2.$$

Since $\hat{H}_1 = \hat{H}_2$*, the Chinese remaindering has stabilized. So we remove the content of* $\hat{H}_2$ *with respect to* x *and get*

$$\hat{G} = (92y^2 - 513z)x^2 + (212y^2 + yz^2 + 125z)x + (251y^2z^2 - 43z^3 + 5y^2 + 318).$$

We test $\hat{G}|A$ *and* $\hat{G}|B$*. Since this is the case, we stop and conclude* $\hat{G} = G$*.*

Remark 2.2. Since the leading term of A or B and the evaluated scaling factor modulo primes never vanish in the computation, the degrees of the univariate GCD images in x must be greater than or equal to the degree of G in x. Since A and B are assumed to be primitive in x, $\hat{G}|A$ or $\hat{G}|B$ implies that $\hat{G}$ is not only a divisor of A or B but also the one with the highest degree in x. Hence $\hat{G}$ is the GCD.

Remark 2.3. We name the loop to compute univariate GCD images as *the GCD iteration* in the thesis. In this example each GCD iteration computes two univariate GCD images. It is possible that the stable feedback polynomial is not the correct one we need because it could stabilize too early. This uncertainty complicates the algorithm. We discuss it in the next section.

2.5 Determining t

The original Ben-Or/Tiwari interpolation solves a Hankel linear system to determine $\Lambda(z)$. See Equation (2.2). But the Ben-Or/Tiwari interpolation itself was derived from the BCH decoding, and therefore the Berlekamp Massey algorithm [Massey, 1969] which was designed to decode BCH codes should be considered.

Berlekamp [Berlekamp, 1968] first developed an algorithm for decoding binary BCH codes in 1968 by solving the *key equation*. See [Berlekamp, 1968] for the definition of the key equation. In 1969, Massey simplified the key equation and provided a variation of

Berlekamp's algorithm, which is called *the Berlekamp-Massey algorithm* or BMA. This algorithm is widely used as a very efficient method to compute the inverse of a matrix with constant diagonals. The Hankel matrix is such an example. But the Berlekamp-Massey algorithm is regarded as very difficult to understand. Several attempts to make the Berlekamp-Massey algorithm more readable include Imamura and Yoshida [Imamura and Yoshida, 1987], Henkel [Henkel, 1992], Jonckheere and Ma [Jonckheere and Ma, 1989]. In 1975, Sugiyama, et al, designed a new algorithm [Sugiyama et al., 1975] to compute the key equation for decoding Goppa codes by using the Euclidean algorithm. Based on the equivalence of the Berlekamp-Massey algorithm and the Euclidean algorithm, Dornstetter [Dornstetter, 1987] designed an algorithm to decode a BCH code based on the Sugiyama method. For the Sugiyama algorithm, we recommend readers to check [Sala, 2009, Pages 62-65]. Sometimes people call the Sugiyama algorithm as the Euclidean Berlekamp-Massey algorithm. Although in most cases the Berlekamp-Massey algorithm is faster than the Sugiyama algorithm, the latter is preferred in coding theory textbooks due to its simpler structure. The Berlekamp-Massey algorithm is presented in Figure 2.2.

Berlekamp-Massey (BMA)

Input: a sequence $[m_0, m_2, \ldots, m_k]$ where $m_i \in \mathbb{Z}_p$ for $0 \leq i \leq k$.

Output: $c(z) = c_L z^L + \cdots + c_1 z + 1$ $(0 \leq L \leq k)$ so that $m_j + \sum_{n=1}^{L} c_n m_{j-n} = 0$ for $L \leq j \leq k$.

1 $c(z) \leftarrow 1$, $B(z) \leftarrow 1$, $x \leftarrow 1$, $L \leftarrow 0$ and $b \leftarrow 1$.

2 for N from 0 to k do

3 Compute the discrepancy

$$\delta \leftarrow m_N + \sum_{i=1}^{L} c_i m_{N-i}.$$

4 Case 1: If $\delta = 0$ then $x \leftarrow x + 1$ and go to step 5.

5 Case 2: If $\delta \neq 0$ and $2L > N$ then
$c(z) \leftarrow c(z) - \delta \cdot b^{-1} z^x B(z), \quad x \leftarrow x + 1.$

6 Case 3: If $\delta \neq 0$ and $2L \leq N$ then
$T(z) \leftarrow c(z), \quad c(z) \leftarrow c(z) - \delta \cdot b^{-1} z^x B(z),$
$L \leftarrow N + 1 - L, \quad B(z) \leftarrow T(z), \quad b \leftarrow \delta, \quad x \leftarrow 1.$

7 Return $c(z)$.

Figure 2.2: Berlekamp Massey algorithm

As we saw in Example 2.4, we run the BMA with a longer input sequence in every GCD iteration until the feedback polynomials $c(z)$ do not change. For two consecutive GCD

iterations, the calculations performed in both BMA calls are identical until the second call encounters those two newly added points. Therefore we can save the final states of all parameters in the first call and just run the BMA for those two new points with the parameters from the first call to avoid re-calculation. We call this version *the incremental Berlekamp-Massey algorithm*. The Sugiyama algorithm runs the Euclidean algorithm on inputs $S(z)$ and z^{2k} where $S(z) = v_0 + v_1 z + \cdots + v_{2k-1} z^{2k-1}$ for $k = 1, 2, \ldots, t, t+1$. There is no obvious incremental implementation because it builds the feedback polynomial from higher degree to lower degree as more points are appended to the end of the input sequence. On the other hand the BMA builds the result from lower to higher degree. Our GCD algorithm calls BMA at least t times. The incremental version reduces total cost of BMA from $O(t^3)$ to $O(t^2)$ arithmetic operations in the coefficient field $\mathbb{Z}_p$.

In most articles, when using Ben-Or/Tiwari interpolation, authors usually use the Berlekamp-Massey algorithm to solve the Hankel matrix without explanation. In this section we first explore why can the Berlekamp-Massery algorithm determine the unique solution of a Hankel matrix.

We follow the definition of LFSR in [Berlekamp, 1968]. The *linear feedback shift register* (LFSR) is represented by the pair $(L, c(z))$ where

$$c(z) = 1 + c_1 x + c_2 x^2 + \cdots + c_L x^L$$

is the *feedback polynomial* and L is the *length* of the LFSR. A LFSR is said to generate a finite sequence $b = [b_0, b_1, \ldots, b_{N-1}]$ when this sequence coincides with the first N outputs of the LFSR for some initial loading. If $L \geq N$, then the LFSR always generates the sequence. If $L < N$, the LFSR generates the sequence if and only if

$$b_k + b_{k-1} c_1 + b_{k-2} c_2 + \cdots + b_{k-L} c_L = 0,$$

for all $L \leq k \leq N - 1$.

In BCH decoding theory, $c(z)$ is also called the *classical error locator polynomial*. In our algorithm $\Lambda(z)$ is defined to be the reciprocal of $c(z)$. In some articles $\Lambda(z)$ is called the *plain error locator polynomial*. The Berlekamp-Massey algorithm computes the classical one. Hence we need to convert it to the plain version. We mention that the degree of $c(z)$ could be lower than L.

Example 2.5. *Consider the sequence* $a = [17, 10, 55, 64, 2, 50, 43, 29, 69, 72]$ *and* $p = 73$*. Its corresponding Hankel matrix is*

$$H = \begin{bmatrix} 17 & 10 & 55 & 64 & 2 \\ 10 & 55 & 64 & 2 & 50 \\ 55 & 64 & 2 & 50 & 43 \\ 64 & 2 & 50 & 43 & 29 \\ 2 & 50 & 43 & 29 & 69 \end{bmatrix}$$

Note H *is anti-symmetric. The rank of* H *modulo* p *is 5, so* H *is non-singular. If we run the Berlekamp Massey algorithm on* a*, we find* $L = 5$ *and*

$$c(z) = 52z^4 + 68z^3 + 15z^2 + 66z + 1$$

in which the leading coefficient vanishes modulo p*, therefore we have* $\deg_z c(z) = 4 < 5 = L$*.*

Therefore the rank of the Hankel matrix is equal to the length L, not the degree of $c(z)$. Let $(L(n), c^n(z))$ denote a *shortest* LFSR that generates the sequence b. BMA computes a shortest LFSR for a given sequence b. If the Hankel matrix generated by b is non-singular then the following lemma guarantees $L(r) = r/2$.

Lemma 2.5. *[Imamura and Yoshida, 1987] Let* r *be a positive even integer,* $b = [b_0, \ldots, b_{r-1}]$ *be a sequence and* $L(r)$ *be the shortest length among all LFSR generating* b*. Then* $L(r) = r/2$ *if and only if the Hankel matrix generated by* $[b_0, \ldots, b_{r-1}]$ *is non-singular.*

Massey [Massey, 1969] observed that if the BMA terminates with $2L(r) > r$ then the LFSR with the shortest length is not unique. However if the resulting LFSR satisfies $2L(r) \leq r$ then uniqueness is guaranteed.

Theorem 2.3. *[Massey, 1969, Theorem 3] Let* $(L(r), c^r(z))$ *be the output of the Berlekamp-Massey algorithm on input* $v = [v_0, v_1, \ldots, v_{r-1}]$*. If* $2L(r) \leq r$*, then the* $(L(r), c^r(z))$ *is the unique shortest LFSR.*

Now we explain how does the BMA solve a Hankel system in our case. For example, the case shown in Example 2.5 is never going to happen. Let $f = \sum_{i=1}^{t} a_i M_i \in \mathbb{Z}_p[x_1, \ldots, x_n]$ where p is a prime. We also assume $Supp(f) = Supp(f \mod p)$. Suppose we evaluate f at $2t$ consecutive points required by the discrete logarithm method and get $v_i = f(w_1^i, w_2^i, \ldots, w_n^i)$ for $0 \leq i \leq 2t-1$. Let $m_i = M_i(w_1^i, w_2^i, \ldots, w_n^i)$ for $1 \leq i \leq t$. Let us decompose the $t \times t$ Hankel matrix H_t generated by the sequence $[v_0, \ldots, v_{2t-1}]$. The following decomposition can be found in [Ben-Or and Tiwari, 1988] and [E Kaltofen and Lobo, 2000].

$$H_t = \begin{bmatrix} v_0 & v_1 & \cdots & v_{t-2} & v_{t-1} \\ v_1 & v_2 & \cdots & v_{t-1} & v_t \\ v_2 & v_3 & \cdots & v_t & v_{t+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ v_{t-1} & v_t & \cdots & v_{2t-3} & v_{2t-2} \end{bmatrix} = V \times Q \times V^T \text{ where}$$

$$V = \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 \\ m_1 & m_2 & \cdots & m_{t-1} & m_t \\ m_1^2 & m_2^2 & \cdots & m_{t-1}^2 & m_t^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m_1^{t-1} & m_2^{t-1} & \cdots & m_{t-1}^{t-1} & m_t^{t-1} \end{bmatrix} \quad \text{and} \quad Q = \begin{bmatrix} a_1 & 0 & \cdots & 0 & 0 \\ 0 & a_2 & \cdots & 0 & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & a_t \end{bmatrix}.$$

Now $Supp(f) = Supp(f \mod p)$ implies that the determinant of Q is non-zero. The determinant of V is $\prod_{0 \le i < j \le t-1}(m_i - m_j)$ which is non-zero since $m_i \neq m_j$ if $i \neq j$ by our choice of p and Proposition 2.1. Therefore the determinant of H_t is non-zero and H_t is non-singular. If we run the BMA on $[v_0, \ldots, v_{2t-1}]$ then $L(2t) = t$ by Lemma 2.5. Next we want to show that the leading coefficient of $c(z)$ returned by BMA cannot vanish modulo p. The polynomial $\Lambda(z)$ has degree t with the constant term $\prod_{i=1}^{t} m_i$ which is not zero modulo p. Hence the reciprocal of Λ, which is $c(z)$, has degree t with the leading coefficient $\prod_{i=1}^{t} m_i \mod p$ which is non-zero and $L(2t) = \deg_z c(z)$. Therefore if we run the BMA on $2t$ consecutive evaluation points, then $\deg_z c(z) = t$ and its reciprocal must factor to t distinct linear factors. $L(2t) = t$ also ensures the uniqueness of $c(z)$ by Theorem 2.3.

Ben-Or and Tiwari observed that the Hankel matrix $H(k, k)$ must be singular if $k > t$. Unfortunately, this can be true for $k < t$ too. We constructed the following example by forcing some discrepancies to be zero so that the feedback polynomial does not change.

Example 2.6. *Consider the polynomial*

$$f(y) = y^7 + 60y^6 + 40y^5 + 48y^4 + 23y^3 + 45y^2 + 75y + 55$$

with coefficients in the finite field $\mathbb{Z}_{101}$. *Let* $\omega = 93$ *be a generator of* $\mathbb{Z}_{101}^*$. *Since* f *has 8 terms,* 16 *points are required to determine the correct* $\Lambda(z)$ *and two more are needed for confirmation. We compute*

$$(f(\omega^j))_{0 \le j \le 17} = v = [44, 95, 5, 51, 2, 72, 47, 44, 21, 59, 53, 29, 71, 39, 2, 27, 100, 20].$$

We run the Berlekamp-Massey algorithm on input $v_r = [v_0, \ldots v_{2r-1}]$ *for* $1 \le r \le 9$ *and obtain feedback polynomials in the following table.*

r	*Output* $c(z)$
1	$69z + 1$
2	$24z^2 + 59z + 1$
3	$24z^2 + 59z + 1$
4	$24z^2 + 59z + 1$
5	$70z^7 + 42z^6 + 6z^3 + 64z^2 + 34z + 1$
6	$70z^7 + 42z^6 + 25z^5 + 87z^4 + 16z^3 + 20z^2 + 34z + 1$
7	$z^7 + 67z^6 + 95z^5 + 2z^4 + 16z^3 + 20z^2 + 34z + 1$
8	$31z^8 + 61z^7 + 91z^6 + 84z^5 + 15z^4 + 7z^3 + 35z^2 + 79z + 1$
9	$31z^8 + 61z^7 + 91z^6 + 84z^5 + 15z^4 + 7z^3 + 35z^2 + 79z + 1$

The ninth call of the BMA confirms that the feedback polynomial returned by the eighth call is the desired one. But, by our design, the algorithm terminates at the third call because the feedback polynomial remains unchanged from the second call. In this case, $\lambda(z) = z^2 c(1/z) = z^2 + 59z + 24$ *has roots* 56 *and* 87 *which correspond to monomials* y^4 *and* y^{20} *respectively.*

The feedback polynomial shown in the above example stabilizes too early and wrong monomials are introduced to the support. For instance, y^{20} in the above example is not a monomial in $Supp(f)$.

In the above example, for $1 \leq k \leq 9$, the k-th call to the BMA performs two more iterations than the $(k-1)$-th call to the BMA because two more points are added. The connection polynomial in the k-th call to the BMA showed in the table of Example 2.6 is the result of the $2k$-th (the last) iteration in each BMA call, and therefore we do not know what happened to each iteration inside the BMA. As shown in Figure 2.2, with a non-zero discrepancy ($\delta \neq 0$), the BMA uses case 3 to increase the degree of the feedback polynomial (the length of LFSR) and case 2 to adjust the coefficients of the feedback polynomial. Hence the BMA iterates between: case 3, case 2, case 3, case 2, ..., etc.

Example 2.7. *Consider* $f = 23x^2 + 11x + 6$ *and* $p = 101$. *We pick* $\omega = 2$ *as the generator of* $\mathbb{Z}_p^*$. $[f(w^i) \mod p : 0 \leq i \leq 7] = [40, 19, 14, 51, 10, 74, 79, 1]$. *We run BMA on this sequence and output every intermediate* $c(z)$ *and its corresponding case number in each iteration.*

i	δ	*Case*	$c(z)$
0	40	3	$1 + 61z$
1	35	2	$1 + 98z$
2	58	3	$44z^2 + 98z + 1$
3	37	2	$79z^2 + 19z + 1$
4	65	3	$26z^3 + 3z^2 + 19z + 1$
5	74	2	$93z^3 + 14z^2 + 94z + 1$
6	0	1	$93z^3 + 14z^2 + 94z + 1$
7	0	1	$93z^3 + 14z^2 + 94z + 1$

Since $\delta = 0$ at iteration 6 and iteration 7, we are in case 1 and $c(z)$ does not change. For non-zero δ, the degree of $c(z)$ increases at iteration $2k$. The coefficients of $c(z)$ are corrected at iteration $2k+1$. This is the general pattern for one call to the BMA.

Now we discuss the termination condition we propose to use in our GCD algorithm. Suppose $H_j = \sum h_{j,i}x^i$ where $h_{j,i} \in \mathbb{Z}_p$ is the j-th scaled univariate GCD image in a GCD iteration. As we saw in Example 2.4, the GCD iteration continues if any feedback polynomial is not stable. Every iteration computes two more univariate GCD images, say H_k and H_{k+1}. Hence we can extract two new integer coefficients $h_{k,i}$ and $h_{k+1,i}$ for x^i and append them to the input sequence for the BMA. If all feedback polynomials $c(z)$ stabilize then we terminate the GCD iteration. Kaltofen [E Kaltofen and Lobo, 2000] pointed out that if $\delta = 0$ and $2L \leq N$, it is likely $\#f$ has been obtained and therefore the GCD iteration can be terminated. Note that Kaltofen's iteration starts from $N = 1$ instead $N = 0$ because he wants to avoid the case $f(\beta_1^0, \ldots, \beta_n^0) = 0 \mod p$ where $\beta_i \in \mathbb{Z}_p$. In this case, $\delta = 0$ at the 0-th iteration which leads to an immediate termination. But for now, we still start the iteration from $N = 0$ and assume that the first discrepancy is non-zero.

In our algorithm we propose to use a double discrepancy test instead of Kaltofen's single test. Suppose there is no zero discrepancy in one call to the BMA on input $[v_0, \ldots, v_{2k-1}]$. After adding two new points v_{2k}, v_{2k+1} to the input sequence, we run the BMA on the new sequence $[v_0, \ldots, v_{2k+1}]$. If both discrepancies δ are zero at iteration $2k$ and $2k+1$, which leads us to case 1 two consecutive times, then the length L does not increase and the coefficients of $c(z)$ are not updated. Hence we conclude that the number of non-zero terms is likely determined. It can be shown that this test is equivalent to comparing feedback polynomials in two consecutive iterations. But checking $\delta = 0$ is cheaper than doing the feedback polynomial coefficients comparison. In our GCD algorithm, each GCD iteration computes two univariate GCD images and therefore two new points are available before we call the BMA. Most importantly, it is possible that $\delta = 0$ at iteration $2k$ but $\delta \neq 0$ at iteration $2k+1$. And doing the double discrepancies test would catch such cases. It is also possible that $\delta \neq 0$ at iteration $2k$, $\delta = 0$ at iteration $2k+1$ but $\delta \neq 0$ at iteration $2k+2$ in the next call of the BMA. In this case, the GCD iteration should not be terminated either.

Example 2.8. *Consider the GCD problem $f_1, f_2 \in \mathbb{Z}_{101}[x, y]$ and $g = \gcd(f_1, f_2)$ where*

$$g = x + (14y^5 + 60y^4 + 72y^3 + 42y^2 + 6y + 52).$$

We want to determine the number of non-zero terms in the constant term of g. Let $\omega = 2$ be a generator of $\mathbb{Z}_{101}^*$. Since the coefficient of x^0 has 6 terms, we need 12 points get the desired $c(z)$. At the first GCD iteration, we compute $\gcd(f_1(x, \omega^i), f_2(x, \omega^i))$ for $0 \leq i \leq 1$ and extract the coefficients of the constant terms to form the sequence $[44, 95]$. We run BMA on the sequence and obtain $c(z) = 69z+1$. We compute the next two univariate GCD images and extract the constant terms to form the sequence $[44, 95, 5, 97]$. Its corresponding feedback

polynomial is $c(z) = 73z^2 + 48z + 1$. *No* $\delta = 0$ *appears in this run of BMA. In the next GCD iteration, we compute two more GCD images for* $i = 4, 5$ *and extract the coefficients of constant terms to form the sequence* $[44, 95, 5, 97, 29, 43]$. *At iteration* 4, *the discrepancy is*

$$\delta = 29 \cdot 1 + 97 \cdot 43 + 5 \cdot 73 = 0 \mod 101.$$

and $2L = 2 \cdot 2 = 4 \leq 4 = N$. *Since* $\delta = 0$, $c(z)$ *is unchanged. If we don't stop and go to the iteration* 5, *the discrepancy is*

$$\delta = 43 \cdot 1 + 29 \cdot 43 + 97 \cdot 73 \equiv 32 \neq 0 \mod 101.$$

In this example, the double discrepancies test would lead us all the way to the 13*-th iteration which confirms that* $c(z)$ *returned at* 11*-th iteration is the one we need.*

Instead of using the first n primes as evaluation points, Kaltofen used random points from a subset S of a given domain as evaluation points to derive the following bound. See [E Kaltofen and Lobo, 2000] for details.

Theorem 2.4 (KL 2000)**.** *Let* $f(x_1, \ldots, x_n) \in \mathbb{Z}[x_1, \ldots, x_n]$ *be a polynomial of* t *non-zero terms. Let* $\beta_1, \ldots, \beta_n$ *be chosen uniformly at random from a subset* S *of* $\mathbb{Z}$. *If we run the Berlekamp-Massey algorithm on the sequence* $[f(\beta_1^i, \ldots, \beta_n^i) : i \geq 0]$, *then we encounter* $\delta = 0$ *the first time at iteration* $2t$ *with probability greater than or equal to*

$$1 - \frac{t(t+1)(2t+1)\deg(f)}{6|S|}.$$

Our current evaluation points are $(\omega^{\frac{p-1}{q_i}})^j$ for $j = 0, 1, 2, \ldots$ and they are not random once ω is chosen. In our GCD algorithm we always want random evaluation points to reduce the probability to encounter bad and unlucky evaluations, and therefore we introduce a shift value s. We pick $s \in \mathbb{Z}_p$ uniformly at random and use $(\omega^{\frac{p-1}{q_i}})^{s+j}$ as evaluation points for $j = 0, 1, 2, \ldots$. Since s is random, the starting evaluation point $(\omega^{\frac{p-1}{q_i}})^{s+0}$ is random in a subgroup of $\mathbb{Z}_p^*$ of order q_i.

A Kronecker substitution, see Chapter 3 for detail, converts multivariate polynomials in $x_0, x_1, \ldots, x_n$ to bivariate polynomials in x, y where $x = x_0$ and $y = x_1$. That is

$$f(x_0, x_1, \ldots, x_n) \to f(x, y, y^{d+1}, y^{(d+1)^2}, \ldots, y^{(d+1)^{n-1}})$$

where d bounds the degree of each variable in the target GCD. If we apply this Kronecker substitution to our inputs, then we only need to interpolate the variable y. This strategy also simplifies the evaluation point from $(\omega^{\frac{p-1}{q_i}})^s$ to $(\omega^{\frac{p-1}{p-1}})^s = \omega^s$. Hence if s is random then ω^s is random in $\mathbb{Z}_p^*$. But with the shift s, one needs to solve a shifted Vandermonde linear system which is to be discussed in Section 2.7.

In Theorem 2.4, S is a subset of $\mathbb{Z}$ and f is a multivariate polynomial. In our case, $S = \mathbb{Z}_p^*$ and a Kronecker substitution leads us to interpolate univariate polynomials of high degree. Therefore we should derive a bound for our case. We first introduce the following popular lemma.

Proposition 2.2 (Cauchy-Binet formula)**.** *Let A be a $m \times n$ matrix and B be a $n \times m$ matrix. Let $[n]$ denote the set $\{1, 2, \ldots, n\}$ and let $\binom{[n]}{m}$ be the set of all m-combinations of $[n]$. For $S \in \binom{[n]}{m}$, let $A_{[m],S}$ be the sub-matrix of A whose columns are the columns of A at indices from S and $B_{S,[m]}$ be the sub-matrix of B whose rows are the rows of B at indices from S. Then we have*

$$\det(AB) = \sum_{S \in \binom{[n]}{m}} \det(A_{[m],S}) \det(B_{S,[m]}).$$

See [Broida and Williamson, 1989, Section 4.6] for a proof. We notice that $m > n$ implies $\det(AB) = 0$ since $rank(AB) \leq \min\{rank(A), rank(B)\} \leq n$ and AB is a $m \times m$ matrix.

Example 2.9. *In this example, we compute* $\det AB$ *where*

$$A = \begin{bmatrix} 5 & 2 & 1 \\ 3 & 3 & 9 \end{bmatrix} \quad \textit{and} \quad B = \begin{bmatrix} 2 & 5 \\ 8 & 3 \\ 9 & 3 \end{bmatrix}.$$

It is easy to check that

$$\det AB = \det \begin{bmatrix} 35 & 34 \\ 111 & 51 \end{bmatrix} = -1989.$$

Since $[n] = \{1, 2, 3\}$ and $m = 2$, $\binom{[n]}{m} = \{\{1, 2\}, \{1, 3\}, \{2, 3\}\}$. According to Cauchy-Binet formula, we have

$$\begin{aligned} \det AB &= \det \begin{bmatrix} 5 & 2 \\ 3 & 3 \end{bmatrix} \det \begin{bmatrix} 2 & 5 \\ 8 & 3 \end{bmatrix} + \det \begin{bmatrix} 5 & 1 \\ 3 & 9 \end{bmatrix} \det \begin{bmatrix} 2 & 5 \\ 9 & 3 \end{bmatrix} + \det \begin{bmatrix} 2 & 1 \\ 3 & 9 \end{bmatrix} \det \begin{bmatrix} 8 & 3 \\ 9 & 3 \end{bmatrix} \\ &= -306 - 1638 - 45 = -1989. \end{aligned}$$

With a Kronecker substitution, our GCD algorithm interpolates univariate polynomials, and therefore we only consider the univariate interpolation case. Let

$$f(y) = \sum_{i=1}^{t} a_i M_i \in \mathbb{Z}[y] \quad \text{where} \quad M_i = y^{e_i} \text{ and } e_i \in \mathbb{N} \cup \{0\}$$

be the target polynomial to be interpolated. Let p be a prime so that $p \gg \deg_y f$ and ω be a generator of $\mathbb{Z}_p^*$. Our algorithm picks a random $s \in \mathbb{Z}_p \setminus \{p - 1\}$ as a shift and eval-

uates the inputs at $\omega^s, \omega^{s+1}, \ldots$. Now we fix ω and the map $\omega : s \longrightarrow \omega^s$ is a bijection. If s is chosen uniformly at random then ω^s is uniformly random in $\mathbb{Z}_p^*$. Let $\beta = \omega^s$. Then the evaluation points sequence becomes $\beta, \beta\omega, \beta\omega^2, \beta\omega^3, \ldots$. We derive an upper bound for the number of evaluation points so that the Berlekamp-Massey algorithm on input sequence $[f(\beta), f(\beta\omega), \ldots, f(\beta\omega^{2t-1})]$ encounters a zero discrepancy before we get the correct feedback polynomial.

We define $F_i = f(x\omega^i)$ where x is a variable and $M_i(\omega) = m_i$. For example, if $f = 35y^{10} - 20y^5 + 5$ then $F_4 = f(x\omega^4) = 35x^{10}(\omega^{10})^4 - 20x^5(\omega^5)^4 + 5$. Let A_i be an $i \times i$ Hankel matrix

$$A_i = \begin{bmatrix} F_0 & F_1 & \cdots & F_{i-2} & F_{i-1} \\ F_1 & F_2 & \cdots & F_{i-1} & F_i \\ F_2 & F_3 & \cdots & F_i & F_{i+1} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ F_{i-1} & F_i & \cdots & F_{2i-3} & F_{2i-2} \end{bmatrix}.$$

The Hankel matrix A_i can be decomposed as $A_i = B_i \times C_t \times B_i^T$ where

$$B_i = \begin{bmatrix} 1 & 1 & \cdots & 1 & 1 \\ m_1 & m_2 & \cdots & m_{t-1} & m_t \\ m_1^2 & m_2^2 & \cdots & m_{t-1}^2 & m_t^2 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m_1^{i-1} & m_2^{i-1} & \cdots & m_{t-1}^{i-1} & m_t^{i-1} \end{bmatrix}, \quad C_t = \begin{bmatrix} a_1M_1(x) & 0 & \cdots & 0 \\ 0 & a_2M_2(x) & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & a_tM_t(x) \end{bmatrix}.$$

Note that B_i is a standard Vandermonde matrix. We first prove the following theorem.

Theorem 2.5. *Symbolically* $\det A_i \neq 0 \in \mathbb{Z}_p[x]$ *for* $1 \leq i \leq t-1$.

Proof. By the Cauchy-Binet formula we have

$$\det(A_i) = \sum_{J \in \binom{[t]}{i}} \det((B_i)_{[i],J}) \det(C_t B_i^T)_{J,[i]}, \tag{2.6}$$

where J is a i-combination of $\{1, 2, \ldots, t\}$. For each J, we have

$$\det(C_t B_i^T)_{J,[i]} = \sum_{K \in \binom{[t]}{i}} \det((C_t)_{J,K}) \det((B_i^T)_{K,[i]}), \tag{2.7}$$

where K is a i-combination of $\{1, 2, \ldots, t\}$. Combining equations (2.6) and (2.7), we have

$$\det(A_i) = \sum_{J \in \binom{[t]}{i}} \sum_{K \in \binom{[t]}{i}} \det((B_i)_{[i],J}) \det((C_t)_{J,K}) \det((B_i^T)_{K,[i]}).$$

Since C_t is a diagonal matrix, $\det((C_t)_{J,K}) = 0$ if $J \neq K$. Therefore the above equation can be simplified to

$$\det(A_i) = \sum_{J \in \binom{[t]}{i}} \det((B_i)_{[i],J}) \det((C_t)_{J,K}) \det((B_i^T)_{K,[i]}).$$

Without loss of generality we assume that monomials are ordered in degree by $M_1 > M_2 > \cdots > M_t$. Let $J = \{1, 2, \ldots, i\}$. Then by the property of the Vandermonde matrix we have

$$\det((B_i)_{[i],J}) \det((C_t)_{J,K}) \det((B_i^T)_{K,[i]}) = a_1 \cdots a_i M_1(x) \cdots M_i(x) (\prod_{1 \leq v < u \leq i} (m_v - m_u))^2.$$

Since $a_k \neq 0$ for $1 \leq k \leq t$ and $m_i \neq m_j \mod p$ if $i \neq j$,

$$a_1 \cdots a_i (\prod_{1 \leq v < u \leq i} (m_v - m_u))^2 \mod p \neq 0.$$

The leading monomial of $\det A_i$ is $M_1(x) \cdots M_i(x)$ and it has a non-zero coefficient in $\mathbb{Z}_p$. Therefore $\det A_i \neq 0$. □

If we use $\beta = \omega^s$ as an evaluation point, then the condition that the Berlekamp-Massey algorithm does not encounter a zero discrepancy is equivalent to $\prod_{i=1}^t \det A_i(\beta) \neq 0 \mod p$. Therefore if β is not a root of $\prod_{i=1}^t \det A_i(x)$ then we must obtain the correct feedback polynomial. We can now bound the probability that we don't obtain a correct feedback polynomial.

Theorem 2.6. *Let f be a univariate polynomial to be interpolated, $\#f = t$, p be a prime and $p \gg \deg_y f$. Let ω be a generator of $\mathbb{Z}_p^*$. Then the number of shift values s which make the Berlekamp-Massey algorithm encounter a zero discrepancy on $[f(\beta), f(\beta\omega), \ldots, f(\beta\omega^{2t-1})]$ is at most $\frac{t(t+1)deg_y f}{2} = \binom{t+1}{2} \deg_y f$ where $\beta = \omega^s$. Therefore if s is chosen uniformly at random from $[0, p-2]$, then the probability that the Berlekamp-Massey algorithm encounters a zero discrepancy for the first time at iteration $2t$ is greater than or equal to $1 - \frac{t(t+1)\deg_y f}{2(p-1)} = 1 - \frac{\binom{t+1}{2}\deg_y f}{p-1}$.*

Proof. Since $\deg_y \det A_i \leq i \deg_y f$, we have

$$\begin{aligned} \deg_y \prod_{i=1}^t \det A_i &= \sum_{i=1}^t \deg_y \det A_i \leq \sum_{i=1}^t i \deg_y f \\ &= \deg_y f \sum_{i=1}^t i = \frac{t(t+1)\deg_y f}{2} = \binom{t+1}{2} \deg_y f. \end{aligned}$$

Therefore the number of roots of $\prod_{i=1}^t \det A_i$ is bounded by $\binom{t+1}{2} \deg_y f$. □

The bound we derived is the worst case. To get a sense of the average number of encountering $\delta = 0$ in the first $2t$ iterations, we wrote a C program to generate all univariate polynomials $f(x) \in \mathbb{Z}_p$ where $\#f = t$, $\deg_x f \leq d$. For each f, we ran the BMA on $[f(\omega^0), f(\omega^1), \ldots, f(\omega^{2t-1})]$ to check whether $\delta = 0$ in any iteration in BMA. The last column of Table 2.5 shows the percentage of encountering $\delta = 0$ among all f generated. As

d	t	ω	prime	total cases	early termination	$\cdot\%\delta = 0$
4	3	3	17	40960	10614	25.9131
4	3	5	17	40960	10855	26.5015
4	3	10	17	40960	10855	26.5014
4	3	5	47	973360	97690	10.0364
5	3	5	47	1946720	197495	10.1450
6	3	5	47	3406760	346189	10.1618
5	3	2	101	20000000	964340	4.8217
6	3	2	101	35000000	1687595	4.8217
4	3	2	211	92610000	2167370	2.3403

Table 2.3: Early termination test

we see in the table, the probability that the BMA encounters $\delta = 0$ drops dramatically as the size of p increases. From Table 2.5, we observe that the percentage of cases which result in early termination is asymptotically equal to C/p where C is some non-zero constant. The size of p or $|S|$ in Theorem 2.6 is a parameter that we can change. We mostly use 63 bits prime or 127 bits prime in our implementation, and therefore the probability to encounter $\delta = 0$ is very low.

For example, one of our second benchmark problems has a target GCD in 9 variables, each variable has degree at most 20 and $t = 10^5$. If we use the Kronecker substitution $K_r(G(x_0, x_1, \ldots, x_8)) = G(x, y, y^{21}, y^{21^2}, \ldots, y^{21^7})$, then $\deg_y K_r(G) \leq 20+20\cdot 21+20\cdot 21^2+\cdots+20\cdot 21^7 = 20\sum_{k=1}^{8} 21^{k-1} = 37822859360$. According to Theorem 2.6, if we use a 127bits prime, then the probability that the BMA encounters a zero discrepancy the first time at $2t$-th evaluation point is greater than or equal to $1-\frac{37822859360\cdot 10^5\cdot(10^5+1)}{2(2^{127}-1)} = 1-1.11\times 10^{-18}$. Hence it is very rare to obtain an incorrect feedback polynomial in our GCD algorithm.

2.6 Bad and unlucky evaluation points

Let A and B be non-constant polynomials in $\mathbb{Z}[x_0, \ldots, x_n]$, $G = \gcd(A, B)$ and $\bar{A} = A/G$ and $\bar{B} = B/G$. Let p be a prime such that $LC(A)LC(B) \mod p \neq 0$.

Definition 2.1. Let $\alpha \in \mathbb{Z}_p^n$ and let $\bar{g}_\alpha(x) = \gcd(\bar{A}(x, \alpha), \bar{B}(x, \alpha))$. We say α is *bad* if $LC(A)(\alpha) = 0$ or $LC(B)(\alpha) = 0$ and α is *unlucky* if $\deg \bar{g}_\alpha(x) > 0$. If α is not bad and not unlucky, we call it *good*.

Example 2.10. *Consider the GCD problem of* $A = G\bar{A}$ *and* $B = G\bar{B}$ *where*

$$G = (x_1 - 16)x_0 + 1, \quad \bar{A} = x_0^2 + 1 \quad \textit{and} \quad \bar{B} = x_0^2 + (x_1 - 1)(x_2 - 9)x_0 + 1.$$

Then $LC(A) = LC(B) = x_1 - 16$. *So* $\{(16, \beta) : \beta \in \mathbb{Z}_p\}$ *are bad.* $\{(1, \beta) : \beta \in \mathbb{Z}_p\}$ *and* $\{(\beta, 9) : \beta \in \mathbb{Z}_p\}$ *are unlucky.*

Our GCD algorithm can not reconstruct G using the image $g(x, \alpha) = \gcd(A(x, \alpha), B(x, \alpha))$ if α is unlucky due to the wrong degree of x in g. Brown's idea in [Brown, 1971] to detect unlucky α is based on the following Lemma.

Lemma 2.6. *Let* α *and* g_α *be as above and* $h_\alpha = G(x, \alpha) \mod p$. *If* α *is not bad then* $h_\alpha | g_\alpha$ *and* $\deg_x g_\alpha \geq \deg_x G$.

See [Geddes et al., 1992, Lemma 7.3] for a proof of Lemma 2.6. Brown only uses α which are not bad and the images $g_\alpha(x)$ of least degree to interpolate G. The following Lemma implies if the prime p is large then unlucky evaluation points are rare.

Lemma 2.7. *If* α *is chosen uniformly at random from* $\mathbb{Z}_p^n$ *then*

$$\text{Prob}[\ \alpha \text{ is bad or unlucky }] \leq \frac{\deg AB + \deg A \deg B}{p}.$$

Proof. Let b be the number of bad evaluation points and let r be the number of unlucky evaluation points that are not bad. Let B denote the event that α is bad and G denote the event that α is not bad and U denote the event α is unlucky. Then

$$\begin{aligned} \text{Prob}[B \text{ or } U] &= \text{Prob}[B] + \text{Prob}[G \text{ and } U] \\ &= \text{Prob}[B] + \text{Prob}[G] \times \text{Prob}[U|G] \\ &= \frac{b}{p^n} + \left(1 - \frac{b}{p^n}\right) \frac{r}{p^n - b} \ = \ \frac{b}{p^n} + \frac{r}{p^n}. \end{aligned}$$

Now α is bad $\Longrightarrow \text{LC}(A)(\alpha)\text{LC}(B)(\alpha) = 0 \Longrightarrow LC(AB)(\alpha) = 0$. Applying Lemma 2.1 with $f = \text{LC}(AB)$ we have $b \leq \deg LC(AB) p^{n-1}$. Let $R = \text{res}_{x_0}(\bar{A}, \bar{B}) \in \mathbb{Z}_p[x_1, \ldots, x_n]$. Now α is unlucky and not bad $\Longrightarrow \deg \gcd(\bar{A}(x, \alpha), \bar{B}(x, \alpha)) > 0$ and $\text{LC}(\bar{A})(\alpha) \neq 0$ and $\text{LC}(\bar{B})(\alpha) \neq 0 \Longrightarrow R(\alpha) = 0$ by Lemma 2.4 (iv) and (v). Applying Lemma 2.1 we have $r \leq \deg(R) p^{n-1}$. Substituting into the above we have

$$\text{Prob}[B \text{ or } U] \leq \frac{\deg \text{LC}(AB)}{p} + \frac{\deg R}{p} \leq \frac{\deg AB}{p} + \frac{\deg A \deg B}{p}.$$

□

The algorithm shown in Figure 2.3 applies Lemma 2.7 to compute an upper bound d for $\deg_{x_i} G$. If *DegreeBound* hits an unlucky evaluation and returns $d > \deg_{x_i} G$, it won't affect the correctness of our algorithm, only the efficiency.

Algorithm DegreeBound(A,B,i)

Input: Non-zero $A, B \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$ and i satisfying $0 \leq i \leq n$.

Output: $d \geq \deg_{x_i}(G)$ where $G = \gcd(A, B)$.

1 Set $LA = \mathrm{LC}(A, x_i)$ and $LB = \mathrm{LC}(B, x_i)$.
So $LA, LB \in \mathbb{Z}[x_0, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n]$.

2 Pick a prime $p \gg \deg A \deg B$ such that $LA \mod p \neq 0$ and $LB \mod p \neq 0$.

3 Pick $\alpha = (\alpha_0, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n) \in \mathbb{Z}_p^n$ uniformly at random until $LA(\alpha)LB(\alpha) \neq 0$.

4 Compute $a = A(\alpha_0, \ldots, \alpha_{i-1}, x_i, \alpha_{i+1}, \ldots, \alpha_n)$ and
$b = B(\alpha_0, \ldots, \alpha_{i-1}, x_i, \alpha_{i+1}, \ldots, \alpha_n)$.

5 Compute $g = \gcd(a, b)$ in $\mathbb{Z}_p[x_i]$ using the Euclidean algorithm and output $d = \deg_{x_i} g$.

Figure 2.3: Degree bound algorithm

2.7 Solving shifted Vandermonde system

The first time we considered to shift the evaluation sequence was motivated by the following example.

Example 2.11. *Let $A = (x+1)(2x+y)$ and $B = (x+1)(2x+2y-1)$. $G = \gcd(A, B) = x+1$. If we evaluate A and B at $y = \beta^0 = 1$, no matter what β we pick, we always get an unlucky result* $\gcd(A(x, \beta^0), B(x, \beta^0)) = (x+1)(2x+1)$. *We cannot use the GCD image evaluated at $\beta^0 = 1$ in this example.*

In Example 2.11, all $\beta \in \mathbb{Z}_p$ are unlucky. Shifting the starting point to β^1 could solve this particular problem. But what if β^1 is still unlucky ? In modular polynomial GCD algorithms we prefer random evaluation points to avoid bad and unlucky points. Therefore instead of shifting only by 1 to get β^1, we shift a random value s where $0 \leq s < p-1$ so that the starting evaluation point is β^s. But shifting the starting point increases the power of each entry in the standard Vandermonde matrix. Hence Zippel's quadratic Vandermonde system solving algorithm needs to be modified. Now suppose we have a shifted Vandermonde system $Wc = u$ where W and u are given and we solve for c over $\mathbb{Z}_p$. Explicitly, we have the linear system

$$Wc = \begin{bmatrix} m_1^s & m_2^s & m_3^s & \ldots & m_t^s \\ m_1^{s+1} & m_2^{s+1} & m_3^{s+1} & \ldots & m_t^{s+1} \\ m_1^{s+2} & m_2^{s+2} & m_3^{s+2} & \ldots & m_t^{s+2} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ m_1^{k+t-1} & m_2^{s+t-1} & m_3^{s+t-1} & \ldots & m_t^{s+t-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_t \end{bmatrix} = \begin{bmatrix} v_s \\ v_{s+1} \\ v_{s+2} \\ \vdots \\ v_{s+t-1} \end{bmatrix} = u.$$

We define a polynomial $P_i(z)$ and expand it

$$P_i(z) = z^s \cdot \prod_{\substack{1 \le j \le t \\ j \ne i}} \frac{z - m_j}{m_i - m_j} \tag{2.8}$$

$$= a_{i,1}z^s + a_{i,2}z^{s+1} + a_{i,3}z^{k+2} + \cdots + a_{i,t}z^{s+t-1} \tag{2.9}$$

where $a_{i,k} \in \mathbb{Z}_p$ for $1 \le k \le t$. It is clear that

$$P_i(m_j) \;=\; \begin{cases} m_i^s & \text{if } i = j, \\ 0 & \text{otherwise.} \end{cases}$$

Let A be a $t \times t$ matrix formed by $a_{i,k}$

$$A = \begin{bmatrix} a_{1,1} & a_{1,2} & a_{1,3} & \dots & a_{1,t} \\ a_{2,1} & a_{2,2} & a_{2,3} & \dots & a_{2,t} \\ a_{3,1} & a_{3,2} & a_{3,3} & \dots & a_{3,t} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ a_{t,1} & a_{t,2} & a_{t,3} & \dots & a_{t,t} \end{bmatrix}.$$

The (i,j)-th element of $A \cdot W$ is

$$a_{i,1}m_j^s + a_{i,2}m_j^{s+1} + a_{i,3}m_j^{s+2} + \cdots + a_{i,t}m_j^{s+t-1}.$$

Therefore we have

$$A \cdot W = \begin{bmatrix} m_1^s & 0 & 0 & \dots & 0 \\ 0 & m_2^s & 0 & \dots & 0 \\ 0 & 0 & m_3^s & \dots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \dots & m_t^s \end{bmatrix}$$

and $W^{-1} = W^{-1} \cdot A^{-1} \cdot A = (A \cdot W)^{-1} \cdot A$. Explicitly we have

$$W^{-1} = \begin{bmatrix} \frac{1}{m_1^s} & 0 & 0 & \cdots & 0 \\ 0 & \frac{1}{m_2^s} & 0 & \cdots & 0 \\ 0 & 0 & \frac{1}{m_3^s} & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & \frac{1}{m_t^s} \end{bmatrix} \cdot A = \begin{bmatrix} \frac{a_{1,1}}{m_1^s} & \frac{a_{1,2}}{m_1^s} & \frac{a_{1,3}}{m_1^s} & \cdots & \frac{a_{1,t}}{m_1^s} \\ \frac{a_{2,1}}{m_2^s} & \frac{a_{2,2}}{m_2^s} & \frac{a_{2,3}}{m_2^s} & \cdots & \frac{a_{2,t}}{m_2^s} \\ \frac{a_{3,1}}{m_3^s} & \frac{a_{3,2}}{m_3^s} & \frac{a_{3,3}}{m_3^s} & \cdots & \frac{a_{3,t}}{m_3^s} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ \frac{a_{t,1}}{m_t^s} & \frac{a_{t,2}}{m_t^s} & \frac{a_{t,3}}{m_t^s} & \cdots & \frac{a_{t,t}}{m_t^s} \end{bmatrix}.$$

The solution to the shifted Vandermonde system is

$$c = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_t \end{bmatrix} = \begin{bmatrix} \frac{a_{1,1}}{m_1^s}v_s + \frac{a_{1,2}}{m_1^s}v_{s+1} + \frac{a_{1,3}}{m_1^s}v_{s+2} + \cdots + \frac{a_{1,t}}{m_1^s}v_{s+t-1} \\ \frac{a_{2,1}}{m_2^s}v_s + \frac{a_{2,2}}{m_2^s}v_{s+1} + \frac{a_{2,3}}{m_2^s}v_{s+2} + \cdots + \frac{a_{2,t}}{m_2^s}v_{s+t-1} \\ \frac{a_{3,1}}{m_3^s}v_s + \frac{a_{3,2}}{m_3^s}v_{s+1} + \frac{a_{3,3}}{m_3^s}v_{s+2} + \cdots + \frac{a_{3,t}}{m_3^s}v_{s+t-1} \\ \vdots \\ \frac{a_{t,1}}{m_n^s}v_s + \frac{a_{t,2}}{m_t^s}v_{s+1} + \frac{a_{t,3}}{m_t^s}v_{s+2} + \cdots + \frac{a_{t,t}}{m_n^s}v_{s+t-1} \end{bmatrix},$$

or

$$c_i = \sum_{j=s}^{s+t-1} \text{coeff}(P_i(z), z^j)\frac{v_j}{m_i^s} \qquad \text{for} \quad 1 \le i \le t,$$

where $\text{coeff}(P_i(z), z^j)$ denotes the coefficient of z^j in $P_i(z)$. Recall that $\Lambda(z) = \prod_{j=1}^{t}(z - m_j)$. We observe that $\frac{\Lambda(z)}{z-m_i} = \prod_{\substack{1\le j\le t \\ j\ne i}}(z - m_j)$ is the numerator of $P_i(z)$ and $\frac{\Lambda(z)}{z-m_i}\Big|_{z=m_i} = \prod_{\substack{1\le j\le t \\ j\ne i}}(m_i - m_j)$ is the denominator of $P_i(z)$. Therefore each $P_i(z)$ can be derived from $\Lambda(z)$ using $O(t)$ arithmetic operations in $\mathbb{Z}_p$. For $1 \le i \le t$, computing m_i^s costs $O(\log s)$ operations and the computational cost to determine the inverse of m_i^s is linear. Therefore $Wc = u$ can be solved in $O(t^2 + t\log s)$ arithmetic operations.

2.8 The normalization problem

In Section 1.9 we reviewed the normalization problem and discussed several possible solutions. Unfortunately none of them work for our case. Consider the following example.

Example 2.12. *Let $A = \bar{A}G$ and $B = \bar{B}G$ where*

$$\begin{aligned} G &= x + 2y^2 + 3y + 1, \\ \bar{A} &= ((y^2 + 104)x + 1)(x + 1), \\ \bar{B} &= ((y^2 + 3)x + 1)(x + 53y + 1). \end{aligned}$$

We use the DegreeBound algorithm in Section 2.6 to compute the degree of G in x. Suppose DegreeBound picks the prime 53 and the evaluation point 10 for y , then

$$\gcd(\bar{A}(x, 10), \bar{B}(x, 10)) = x^2 + 20x + 19.$$

Hence we can assume $\deg_x G = 2$. Note that DegreeBound only computes an upper bound for the degree of G in x. In this example, it picks $p = 53$ and hits an unlucky prime. The actual degree of G in x is 1. But we don't know this so we assume that $\deg_x G = 2$ is correct. Since $\Gamma = 1$, no scaling is required for univariate GCD images. For GCD computation, suppose we choose $p = 101$ and the generator $\omega = 2$. For convenience, let us compute the first 10 univariate GCD images in a row without GCD iterations. Let $g_i = \gcd(A(x, y =$

$\omega^i), B(x, y = \omega^i))$ mod p *and we have*

$$\begin{aligned}
g_0 &= x^2 + 82x + 52, & g_1 &= x^2 + 44x + 31, & g_2 &= x^2 + 61x + 13 \\
g_3 &= x^2 + 49x + 46, & g_4 &= x^2 + 95x + 63, & g_5 &= x^2 + 30x + 43 \\
g_6 &= x^2 + 14x + 24, & g_7 &= x^2 + 21x + 1, & g_8 &= x^2 + 46x + 57 \\
g_9 &= x^2 + 87x + 80.
\end{aligned}$$

The GCD algorithm always assumes the least degree of univariate GCD images is the correct one. In this example, it is always 2 *but* $\deg_x G = 1$. *This is caused by the unlucky prime* 101. *But our GCD algorithm did not detect it, and therefore we assume* 2 *is the least degree. Since the degree returned by the DegreeBound is also* 2, *the algorithm assumes that* $\deg_x g_i = 2$ *is correct. Let us recover the constant term. We set*

$$m = [52, 31, 13, 46, 63, 43, 24, 1, 57, 80]$$

and let m_i *denote the sub-sequence of* m *consisting of the first* $2i$ *elements. We run the BMA on* m_i *for* $i = 1, 2, 3, 4, 5$ *and obtain*

i	*Output* $c(z)$
1	$85z + 1$
2	$11z^2 + 66z + 1$
3	$31z^3 + 65z^2 + 25z + 1$
4	$49z^4 + 30z^3 + 11z^2 + 65z + 1$
5	$63z^5 + 13z^4 + 18z^3 + 46z^2 + 76z + 1$

The constant term of G *has 3 terms. Hence* $c(z)$ *should remain the same for* $i = 3, 4, 5$ *but it does not. If we continue to compute more univariate GCD images, extract the constant terms and append them to* m. *It is likely that running the BMA on input* m_i *for* $i = 1, 2, 3, \ldots$ *always returns* $c(z)$ *with degree* i. *Therefore the algorithm would always try to compute the next* $c(z)$, *and therefore probably never terminate.*

In the above example we note that $LC(\bar{A}) \mod 101 = LC(\bar{B}) \mod 101 = y^2 + 3$. The looping does not terminate because the Berlekamp-Massey algorithm always tries to determine a stable feedback polynomial for the constant term of

$$(x + 2y^2 + 3y + 1)(x - \frac{1}{y^2 + 3}) \mod 101,$$

namely $\frac{1}{y^2+3}$ which is not even a polynomial. This happens because the *Degreebound* algorithm outputs $\deg_x G \leq 2$, so either $\deg_x G = 1$ or $\deg_x G = 2$ could be correct. At the same time the unlucky primes 53 and 101 make us believe that $\deg_x G = 2$ is correct. Therefore the *non-monic* factor $(y^2 + 3)x + 1$ which contributes to the monic univariate GCD images

can not be detected and the scaling factor $\Gamma = 1$ makes us recover a polynomial fraction which can not be done with the BMA. We note that the same problem may be caused by an unlucky Kronecker substitution. In practice with a large prime (63 bits) it is likely that *DegreeBound* always returns the accurate degree of x. Hence an unlucky prime can be detected immediately. But theoretically we still need to consider this bad case.

Let $\Gamma = LC(A)$ or $LC(B)$, whichever has fewer terms. Our solution is to use Γ as the scaling factor. Then assuming p is not bad, $LC(\gcd(A \mod p, B \mod p))$ must divide both $LC(A) \mod p$ and $LC(B) \mod p$. Thus scaling $g_i(x)$ by $LC(A)(\omega^i) \mod p$ or $LC(B)(\omega^i) \mod p$ will always give an image of a polynomial. The downside of this solution is that it likely changes the sparsity of H, increases t and increases the cost to remove the content of H.

Chapter 3

Kronecker substitution

The Kronecker substitution converts inputs from multivariate polynomials in $\mathbb{Z}[x_0, \ldots, x_n]$ to bivariate polynomials $\mathbb{Z}[x, y]$. This conversion may introduce bad or unlucky cases as we see in the modulo operations. It also makes the degree in y exponential in n. We discuss these issues in this chapter.

In Example 2.10, by using the evaluation sequence $\alpha_j = (w_1^j, w_2^j)$ for $0 \leq j \leq 2t-1$, we found that $\alpha_0 = (1, 1)$ is unlucky. In fact, since $w_1^{q_1} = (\omega^{\frac{p-1}{q_1}})^{q_1} \equiv 1 \mod p$, all $\alpha_{q_1}, \alpha_{2q_1}, \alpha_{3q_1}, \ldots$ are unlucky. Shifting the sequence may solve this problem. But if $q_i < 2t$, we simply don't have enough evaluation points. Recall that for a polynomial f, $\#f$ denotes the number of terms in f. For the GCD problem, t may be even larger than $\max\{\#a_i, \#b_i\}$. See Example 1.13.

We can increase q_i by using a larger prime, but it is still a problematic approach because there is no way to know t in advance. This difficulty leads us to consider using a Kronecker substitution. In our case, a Kronecker substitution maps multivariate polynomials in $\mathbb{Z}[x_0, x_1, \ldots, x_n]$ to bivariate polynomials in $\mathbb{Z}[x, y]$. After making the Kronecker substitution, we need to interpolate $H(x, y) = \Delta(x, y)G(x, y)$ where $\deg_y H(x, y)$ will be exponential in n. The evaluation points for y become the powers of $\omega^{\frac{p-1}{p-1}} = \omega$. The order of ω is $p-1$ and normally $p - 1 > 2t$ is not a problem. The starting evaluation point ω^{s+0} is random if s is chosen uniformly at random from $\mathbb{Z}_p$. To make discrete logarithms in $\mathbb{Z}_p$ feasible, we follow Kaltofen [Kaltofen, 2010] and pick $p = 2^k u + 1 > \deg_y H(x, y)$ with u small.

Definition 3.1. Let D be an integral domain and let f be a polynomial in $D[x_0, x_1, \ldots, x_n]$. Let $r \in \mathbb{Z}^{n-1}$ with $r_i > 0$. Let $K_r : D[x_0, x_1, \ldots, x_n] \to D[x, y]$ be the Kronecker substitution $K_r(f) = f(x, y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{n-1}})$.

First we note that Kronecker substitution is a ring homomorphism. Let $d_i = \deg_{x_i} f$ be the partial degrees of f for $1 \leq i \leq n$. Observe that K_r is invertible if $r_i > d_i$ for $1 \leq i \leq n-1$. As we mentioned in Section 2.8, not all such Kronecker substitutions can be used. We consider an example.

Example 3.1. *Consider the following GCD problem in* $\mathbb{Z}[x,y,z]$.

$$G = x + y + z,\ \bar{A} = x^3 - yz,\ \bar{B} = x^2 - y^2.$$

Since $\deg_y G = 1$ *the Kronecker substitution* $K_r(G) = G(x, y, y^2)$ *is invertible. However* $\gcd(K_r(\bar{A}), K_r(\bar{B})) = \gcd(\bar{A}(x,y,y^2), \bar{B}(x,y,y^2)) = \gcd(x^3 - y^3, x^2 - y^2) = x - y$. *If we proceed to interpolate the* $\gcd(K_r(A), K_r(B))$ *we will obtain* $(x-y)K_r(G)$ *in expanded form from which we can not recover* G.

We call such a Kronecker substitution unlucky. Theorem 3.1 below tells us that the number of unlucky Kronecker substitutions is finite. To detect them we will also avoid bad Kronecker substitutions in an analogous way Brown did to detect unlucky evaluation points.

Definition 3.2. Let K_r be a Kronecker substitution.
We say K_r is *bad* if $\deg_x K_r(A) < \deg_{x_0} A$ or $\deg_x K_r(B) < \deg_{x_0} B$
and K_r is *unlucky* if $\deg_x \gcd(K_r(\bar{A}), K_r(\bar{B})) > 0$. If K_r is not bad and not unlucky, we call it *good*.

Proposition 3.1. *Let* $f \in \mathbb{Z}[x_1, \ldots, x_n]$ *be non-zero and* $d_i \geq 0$ *for* $1 \leq i \leq n$. *Let* X *be the number of Kronecker substitutions* K_r *taken from the sequence*

$$[d_1 + k, d_2 + k, \ldots, d_{n-1} + k] \ \text{ for } k = 1, 2, 3, \ldots$$

for which $K_r(f) = 0$. *Then* $X \leq (n-1)\sqrt{2 \deg f}$.

Proof. $K_r(f) = 0 \iff f(y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{n-1}}) = 0$
$\iff f \bmod \langle x_1 - y, x_2 - y^{r_1}, \ldots, x_n - y^{r_1 r_2 \cdots r_{n-1}} \rangle = 0$
$\iff f \bmod \langle x_2 - x_1^{r_1}, x_3 - x_2^{r_2}, \ldots, x_n - x_{n-1}^{r_{n-1}} \rangle = 0.$

Thus X is the number of ideals $I = \langle x_2 - x_1^{r_1}, \ldots, x_n - x_{n-1}^{r_{n-1}} \rangle$ for which $f \bmod I = 0$ with $r_i = d_i + 1, d_i + 2, \ldots$. We prove that $X \leq (n-1)\sqrt{2 \deg f}$ by induction on n.

If $n = 1$ then I is empty so $f \bmod I = f$ and hence $X = 0$ and the proposition holds. For $n = 2$ we have $f(x_1, x_2) \bmod \langle x_2 - x_1^{r_1} \rangle = 0 \implies x_2 - x_1^{r_1} | f$. Now X is maximal when $d_1 = 0$ and $r_1 = 1, 2, 3, \ldots$. We have

$$\textstyle\sum_{r_1=1}^{X} r_1 \leq \deg f \implies X(X+1)/2 \leq \deg f \implies X < \sqrt{2 \deg f}.$$

For $n > 2$ we proceed as follows. Either $x_n - x_{n-1}^{r_{n-1}} | f$ or it doesn't. If not then the polynomial $S = f(x_1, \ldots, x_{n-1}, x_{n-1}^{r_{n-1}})$ is non-zero. For the sub-case $x_n - x_{n-1}^{r_{n-1}} | f$ we obtain at most $\sqrt{2 \deg f}$ such factors of f using the previous argument. For the case $S \neq 0$ we have

$$S \bmod I = 0 \iff S \bmod \langle x_2 - x_1^{r_1}, \ldots, x_{n-2} - x_{n-1}^{r_{n-2}} \rangle = 0$$

Notice that $\deg_{x_i} S = \deg_{x_i} f$ for $1 \le i \le n-2$. Hence, by induction on n, $X < (n-2)\sqrt{2\deg f}$ for this case. Adding the number of unlucky Kronecker substitutions for both cases yields $X \le (n-1)\sqrt{2\deg f}$. □

Theorem 3.1. *Let $A, B \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$ be non-zero, $G = \gcd(A, B)$, $\bar{A} = A/G$ and $B = \bar{B}/G$. Let $d_i \ge \deg_{x_i} G$. Let X be the number of bad and unlucky Kronecker substitutions K_{r_k} from the sequence $r_k = [d_1 + k, d_2 + k, \ldots, d_{n-1} + k]$ where d_i is a non-negative integer and $k = 1, 2, 3, \ldots$ Then*

$$X \le \sqrt{2}(n-1)\left[\sqrt{\deg A} + \sqrt{\deg B} + \sqrt{\deg A \deg B}\right].$$

Proof. Let $LA = \mathrm{LC}(A)$ and $LB = \mathrm{LC}(B)$ be the leading coefficients of A and B in x_0. Then K_r is bad $\iff$ $K_r(LA) = 0$ or $K_r(LB) = 0$. Applying Proposition 3.1, the number of bad Kronecker substitutions is at most

$$(n-1)(\sqrt{2\deg LA} + \sqrt{2\deg LB}) \le (n-1)(\sqrt{2\deg A} + \sqrt{2\deg B}).$$

Now let $R = \mathrm{res}_{x_0}(\bar{A}, \bar{B})$. We will assume K_r is not bad.

$$\begin{aligned} K_r \text{ is unlucky} &\iff \deg_x(\gcd(K_r(\bar{A}), K_r(\bar{B})) > 0 \\ &\iff \mathrm{res}_x(K_r(\bar{A}), K_r(\bar{B})) = 0 \\ &\iff K_r(\mathrm{res}_x(\bar{A}, \bar{B})) = 0 \\ &\iff K_r(R) = 0 \quad (K_r \text{ is not bad}). \end{aligned}$$

By Proposition 3.1, the number of unlucky Kronecker substitutions $\le (n-1)\sqrt{2\deg R} \le (n-1)\sqrt{2\deg A \deg B}$ by Lemma 2.4. Adding the two contributions proves the theorem. □

Theorem 3.1 tells us that the number of unlucky Kronecker substitutions is finite. In our GCD algorithm we can easily identify one of them as follows. After computing $g_i(x) = \gcd(K_r(A)(x, \alpha^{s+i}), K_r(B)(x, \alpha^{s+i}))$ and if $\deg_x g_i > d_0$ then K_r is unlucky or α^{s+i} is unlucky so we try the next Kronecker substitution $r = [r_1 + 1, r_2 + 1, \ldots, r_{n-1} + 1]$ and increase the size of prime by 1 bit.

It is still not obvious that a Kronecker substitution that is not unlucky can be used because it can create a content in y of exponential degree. The following example shows this problem.

Example 3.2. *Consider the following GCD problem in $\mathbb{Z}[x, y, z, w]$.*

$$G = wx^2 + zy, \;\; \bar{A} = ywx + z, \;\; \bar{B} = yzx + w.$$

We have $\Gamma = wy$ *and* $\Delta = y$. *For* $K_r(f) = f(x, y, y^3, y^9)$ *we have*

$$\begin{aligned}\gcd(K_r(A), K_r(B)) &= K_r(G)\gcd(y^{10}x + y^3, y^4x + y^9)\\ &= (y^9x^2 + y^4)y^3 = y^7(y^5x^2 + 1)\end{aligned}$$

One must not try to compute the bivariate polynomial GCD $\gcd(K_r(A), K_r(B))$ directly because the degree of the content of $\gcd(K(A), K(B))$ (y^7 in our example) can be exponential in the number of variables and we cannot compute this efficiently using the Euclidean algorithm. The crucial observation is that if we compute **monic** images $g_j = \gcd(K(A)(x, \alpha^j), K(B)(x, \alpha^j))$ over a finite field, any content is divided out. When we scale by $K_r(\Gamma)(\alpha^j)$ and interpolate y in $K_r(H)$ using sparse interpolation, we recover any content. We obtain $K_r(H) = K_r(\Delta)K_r(G) = y^{10}x^2 + y^5$, then invert K_r to obtain $H = (yw)x^2 + (y^2z)$. Now we can remove the content y from H to obtain G.

3.1 Unlucky primes

Let A, B be polynomials in $\mathbb{Z}[x_0, x_1, \ldots, x_n]$, $G = \gcd(A, B)$, $\bar{A} = A/G$ and $\bar{B} = B/G$. In the introduction we defined the polynomials $\Gamma = \gcd(LC(A), LC(B))$, $\Delta = \Gamma/LC(G)$ and $H = \Delta G$ where $LC(A)$, $LC(B)$ and $LC(G)$ are the leading coefficients of A, B and G in x_0 respectively. Our GCD algorithm will compute H modulo a sequence of primes $p_1, p_2, \ldots$.

Let $K_r : \mathbb{Z}[x_0, x_1, \ldots, x_n] \to \mathbb{Z}[x, y]$ be a Kronecker substitution

$$K_r(f) = f(x, y, y^{r_1}, y^{r_1r_2}, \ldots, y^{r_1r_2\cdots r_{n-1}}) \text{ for some } r_i > 0.$$

Let p be a prime and let $\phi_p : \mathbb{Z}[x, y] \to \mathbb{Z}_p[x, y]$ denote the modular mapping $\phi(A) = A \bmod p$. Our GCD algorithm will compute $\gcd(K_r(A), K_r(B))$ modulo a prime p. Some primes must be avoided.

Example 3.3. *Consider the following GCD problem in* $\mathbb{Z}[x_0, x_1]$ *where* a *and* b *are positive integers.*

$$G = x_0 + b\,x_1 + 1,\ \bar{A} = x_0 + x_1 + a,\ \bar{B} = x_0 + x_1$$

In this example, $\Gamma = 1$ *so* $H = G$. *Since there are only two variables the Kronecker substitution is* $K_r(f) = f(x, y)$ *hence* $K_r(\bar{A}) = x + y + a$, $K_r(\bar{B}) = x + y$. *Notice that* $\gcd(K_r(\bar{A}), K_r(\bar{B})) = 1$ *in* $\mathbb{Z}[x, y]$, *but* $\gcd(\phi_p(K_r(\bar{A})), \phi_p(K_r(\bar{B}))) = x + y$ *for any prime* $p|a$. *Like Brown's modular GCD algorithm in [Brown, 1971], our GCD algorithm must avoid these primes. Notice also that* $\mathrm{Supp}(\phi_p(K_r(G))) \neq \mathrm{Supp}(K_r(G)) = \{x, y, 1\}$ *for any prime* $p|b$. *Like Zippel's sparse modular GCD algorithm in [Zippel, 1979b], we must avoid these primes too.*

If our GCD algorithm were to choose primes from a pre-computed set of primes $S = \{p_1, p_2, \ldots, p_N\}$ then notice that if we replace a in Example 3.3 with $a = \Pi_{i=1}^{N} p_i$ then every

prime would be unlucky. To guarantee that our GCD algorithm will succeed on all inputs we need to bound the number of primes that cannot be used and pick our prime from a sufficiently large set uniformly at random.

Because our algorithm will always choose $r_i > \deg_{x_i} H$, the Kronecker substitution K_r leaves the coefficients of H unchanged. Let *pmin* be the smallest prime in S. Recall that $H = \sum_{i=0}^{dG} h_i x_0^i$ with $t = \max(\#h_i)$, so we have $\#H \leq (d+1)t$ hence if p is chosen uniformly at random from S then

$$\text{Prob}[\text{Supp}(\phi_p(H)) \neq \text{Supp}(H)] \leq \frac{(d+1)t \log_{pmin} \|H\|}{N}.$$

Theorem 3.2 below bounds $\|H\|$ from the inputs A and B.

Definition 3.3. Let p be a prime and let K_r be a Kronecker substitution. We say p is *bad* if $\deg_x \phi_p(K_r(A)) < \deg_x K_r(A)$ or $\deg_x \phi_p(K_r(B)) < \deg_x K_r(B)$ and p is *unlucky* if $\deg_x \gcd(\phi_p(\bar{K}_r(A)), \phi_p(\bar{K}_r(B))) > 0$. If p is not bad and not unlucky, we call it *good*.

Let $R = \text{res}_x(\bar{A}, \bar{B}) \in \mathbb{Z}[x_1, \ldots, x_n]$ be the Sylvester resultant of $\bar{A}$ and $\bar{B}$. Unlucky primes are characterized as follows; if p is not bad then Lemma 2.4(vii) implies p is unlucky $\iff \phi_p(K_r(R)) = 0$. Unlucky primes are detected using the same approach as described for unlucky evaluations in section 2.6 which requires that we also avoid bad primes. If p is bad or unlucky then p must divide the integer $M = \|K_r(LC(A))\| \cdot \|K_r(LC(B))\| \cdot \|K_r(R)\|$. Let $pmin = \min_{i=1}^{N} p_i$. Thus if p is chosen uniformly at random from S then

$$\text{Prob}[\ p \textit{ is bad or unlucky }] \leq \frac{\log_{pmin} M}{N}.$$

Theorem 3.2 below bounds also M for given inputs A and B.

Proposition 3.2. *Let A be an $m \times m$ matrix with entries $A_{i,j} \in \mathbb{Z}[x_1, x_2, \ldots, x_n]$ satisfying the term bound $\#A_{i,j} \leq t$, the degree bound $\deg_{x_k} A_{i,j} \leq d$ and the coefficient bound $\|A_{i,j}\| < h$ (for $1 \leq i, j \leq m$). Note if a term bound for $\#A_{i,j}$ is not known we may use $t = (1+d)^n$. Let $K_r : \mathbb{Z}[x_1, x_2, \ldots, x_n] \to \mathbb{Z}[y]$ be the Kronecker map $K_r(f) = f(y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{n-1}})$ for $r_k > 0$ and let $B = K_r(A)$ be the $m \times m$ matrix of polynomials in $\mathbb{Z}[y]$ with $B_{i,j} = K_r(A_{i,j})$ for $1 \leq i, j \leq m$. Then*

(i) $\|\det A\| < m^{m/2} t^m h^m$ *and*
(ii) $\|\det B\| < m^{m/2} t^m h^m$.

Proof. To prove (i), let S be the $m \times m$ matrices of integers given by $S_{i,j} = \|A_{i,j}\|_1$. We claim $\|\det A\| \leq H(S)$ where $H(S)$ is Hadamard's bound on $|\det S|$. After applying the Hadamard's bound to S we have

$$H(S) = \prod_{i=1}^{m} \sqrt{\sum_{j=1}^{m} S_{i,j}^2} = \prod_{i=1}^{m} \sqrt{\sum_{j=1}^{m} \|A_{i,j}\|_1^2} < \prod_{i=1}^{m} \sqrt{m(th)^2} = m^{m/2} t^m h^m$$

which establishes (i). To prove the claim, let K_s be a Kronecker substitution with $s_i > md$ and let C be the $m \times m$ matrix with $C_{i,j} = K_r(A_{i,j})$. We note that $\deg_{x_k} A_{i,j} \leq d$ implies that $\deg_{x_k} A \leq md$ for $1 \leq k \leq n$. Hence K_s is a bijective map on the monomials of $\det A$ thus $K_s(\det A) = \det C$ since the Kronecker substitution is a homomorphism. Hence $\|\det A\| = \|\det C\|$. Now let W be the $m \times m$ matrix with $W_{i,j} = \|C_{i,j}\|_1$ and let $H(W)$ be the Hadamard's bound on $|\det W|$. Then $\|\det C\| \leq H(W)$ and since K_r is bijective $S = W$ hence $H(S) = H(W)$. Therefore we have $\|\det A\| = \|\det C\| \leq H(W) = H(S)$ which proves the claim.

To prove (ii), Let S and T be the $m \times m$ matrices of integers given by $S_{i,j} = \|A_{i,j}\|_1$ and $T_{i,j} = \|B_{i,j}\|_1$ for $1 \leq i, j \leq m$. From the claim in part (i) if $r_k > md$ we have $\|\det A\| = \|\det B\| \leq H(T) = H(S)$. Now if $r_k \leq md$ for any $1 \leq k \leq n-1$ then $K_r(\det A)$ is not necessarily one-to-one on the monomials in $\det A$. However, for all $r_k > 0$ and the definition of one-norm we still have

$$\|K_r(A_{i,j})\|_1 \leq \|A_{i,j}\|_1 \text{ for } 1 \leq i, j \leq m$$

so that $T_{i,j} \leq S_{i,j}$ hence $H(T) \leq H(S)$. We have $\|\det B\| \leq H(T) \leq H(S)$ and (ii) follows. □

Theorem 3.2. *Let $A, B, G, \bar{A}, \bar{B}, \Delta, H$ be as given at the beginning of this section and let $R = \operatorname{res}_{x_0}(\bar{A}, \bar{B})$. Suppose $A = \sum_{i=0}^{d_A} a_i(x_1, \ldots, x_n)x_0^i$ and $B = \sum_{i=0}^{d_B} b_i(x_1, \ldots, x_n)x_0^i$ satisfy $\deg A \leq d$, $\deg B \leq d$, $d_A > 0$, $d_B > 0$, $\|a_i\| < h$ and $\|b_i\| < h$. Let $K_r : \mathbb{Z}[x_0, x_1, \ldots, x_n] \to \mathbb{Z}[x, y]$ be the Kronecker map $K_r(f) = f(x, y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{n-1}})$. If K_r is not bad, that is, $K_r(a_{dA}) \neq 0$ and $K_r(a_{dB}) \neq 0$, then*

(i) $\|K_r(LC(A))\| \leq (1+d)^n h$ and $\|K_r(LC(B))\| \leq (1+d)^n h$,
(ii) $\|K_r(R)\| \leq m^{m/2}(1+d)^{nm}E^m$ and
(iii) if $r_i > \deg_{x_i} H$ for $1 \leq i \leq n-1$ then $\|H\| \leq (1+d)^n E^2$

where $m = d_A + d_B$ and $E = \mathrm{e}^{(n+1)d}h$.

Proof. Since $LC(A) \in \mathbb{Z}[x_1, \ldots, x_n]$ we have $\#LC(A) \leq (1+d)^n$ thus $\|K_r(LC(A))\| \leq (1+d)^n \|LC(A)\| \leq (1+d)^n h$. Using the same argument we have $\|K_r(LC(B))\| \leq (1+d)^n h$ which proves (i).

Let $\bar{A} = \sum_{i=0}^{d\bar{A}} \bar{a}_i x_0^i$ and $\bar{B} = \sum_{i=0}^{d\bar{B}} \bar{b}_i x_0^i$. Because $A = G\bar{A}$ and $B = G\bar{B}$, Lemma 2.2 implies $\|\bar{A}\| < E$ and $\|\bar{B}\| < E$. Let S be Sylvester's matrix formed from $K_r(\bar{a}_i)$ and $K_r(\bar{b}_i)$. Now $K_r(R) = \det S$ and S has dimension $d\bar{A} + d\bar{B} \leq dA + dB = m$. Applying Proposition 3.2 to S we have

$$\|K_r(R)\| = \|\det S\| \leq t^m E^m m^{m/2}$$

where $t = \max_{i,j} \#S_{i,j}$. Since $\bar{A}|A$ and $\bar{B}|B$, we have

$$\deg_{x_j} \bar{a}_i(x_1, \ldots, x_n) \leq d \quad \text{and} \quad \deg_{x_j} \bar{b}_i(x_1, \ldots, x_n) \leq d$$

thus $\#S_{i,j} \leq (1+d)^n$ and (ii) follows.

For (iii) since $G|A$ and $\Delta|LC(A)$, Lemma 2.2 implies $\|G\| < E$ and $\|\Delta\| < E$. Thus

$$\|H\| = \|\Delta G\| \leq \#\Delta \cdot \|\Delta\| \cdot \|G\| \leq (1+d)^n E^2.$$

□

Our definition for unlucky primes differs from Brown [Brown, 1971]. Brown's definition depends on the vector degree whereas ours depends only on the degree in x_0 the main variable. The following example illustrates the difference.

Example 3.4. *Consider the following GCD problem*

$$G = x + y + 1,\ \bar{A} = (y+p)x^2 + y^2,\ \bar{B} = yx^3 + y + p,$$

where p is a prime. We have $H = \gcd(\phi_p(\bar{A}), \phi_p(\bar{B})) = \gcd(yx + y^2, yx^2 + y) = y$. Thus by Definition 3.3, p is not unlucky but by Brown's definition, p is unlucky.

Our GCD algorithm in $\mathbb{Z}[x_0, x_1, \ldots, x_n]$ only needs monic images in $\mathbb{Z}_p[x_0]$ to recover H whereas Brown needs monic images in $\mathbb{Z}_p[x_0, x_1, \ldots, x_n]$ to recover G. A consequence of this is that our bound on the number of unlucky primes is much smaller than Brown's [Brown, 1971, Theorems 1 and 2].

3.2 The number of unlucky evaluation points

Even if the Kronecker substitution is not unlucky, after applying it to input polynomials A and B, because the degree in y may be very large, the number of bad and unlucky evaluation points may be very large.

Example 3.5. *Consider the following GCD problem*

$$\begin{aligned} G &= x_0 + x_1^d + x_2^d + \cdots + x_n^d, \\ \bar{A} &= x_0 + x_1 + \cdots + x_{n-1} + x_n^{d+1}, \text{ and} \\ \bar{B} &= x_0 + x_1 + \cdots + x_{n-1} + 1. \end{aligned}$$

To recover G, if we use $r = [d+1, d+1, \ldots, d+1]$ for $x_1, x_2, \ldots, x_{n-1}$ we need $p > (d+1)^n$. But $R = \mathrm{res}_{x_0}(\bar{A}, \bar{B}) = 1 - x_n^{d+1}$ and $K_r(R) = 1 - (y^{r_1 r_2 \cdots r_{n-1}})^{d+1} = 1 - y^{(d+1)^n}$ which means there could be as many as $(d+1)^n$ unlucky evaluation points. If $p = (d+1)^n + 1$, all evaluation points would be unlucky.

To guarantee that we avoid unlucky evaluation points with high probability we would need to pick $p \gg \deg K_r(R)$ which could be much larger than what is needed to interpolate $K_r(H)$. But this upper bound based on the resultant is a worst case. This leads us to investigate what the expected number of unlucky evaluation points is. We ran an experiment.

We computed all monic quadratic and cubic bivariate polynomials over small finite fields $\mathbb{F}_q$ of size $q = 2, 3, 4, 5, 7, 8, 11$ and counted the number of unlucky evaluation points. We generalized our observation to the following result.

Theorem 3.3. *Let $\mathbb{F}_q$ be a finite field with q elements and $f = x^l + \sum_{i=0}^{l-1}(\sum_{j=0}^{d_i} a_{ij}y^j)x^i$ and $g = x^m + \sum_{i=0}^{m-1}(\sum_{j=0}^{e_i} b_{ij}y^j)x^i$ with $l \geq 1$, $m \geq 1$, and $a_{ij}, b_{ij} \in \mathbb{F}_q$. Let $X = |\{\alpha \in \mathbb{F}_q : \gcd(f(x,\alpha), g(x,\alpha)) \neq 1\}|$ be a random variable over all choices $a_{ij}, b_{ij} \in \mathbb{F}_q$. So $0 \leq X \leq q$ and for f and g not coprime in $\mathbb{F}_q[x,y]$ we have $X = q$. If $d_i \geq 0$ and $e_i \geq 0$ then $\mathrm{E}[X] = 1$.*

Proof. Let $C(y) = \sum_{i=0}^{d} c_i y^i$ with $d \geq 0$ and $c_i \in \mathbb{F}_q$ and fix $\beta \in \mathbb{F}_q$. Consider the evaluation map $C_\beta : \mathbb{F}_q^{d+1} \to \mathbb{F}_q$ given by $C_\beta(c_0, \ldots, c_d) = \sum_{i=0}^{d} c_i\beta^i$. We claim that C is balanced, that is, C maps q^d inputs to each element of $\mathbb{F}_q$. It follows that $f(x,\beta)$ is also balanced, that is, over all choices for $a_{i,j}$ each monic polynomial in $\mathbb{F}_q[x]$ of degree n is obtained equally often. Similarly for $g(x,\beta)$.

Recall that two univariate polynomials a, b in $\mathbb{F}_q[x]$ with degree $\deg a > 0$ and $\deg b > 0$ are coprime with probability $1 - 1/q$. See Mullen and Panario [Mullen and Panario, 2013, Chapter 11]). This is also true under the restriction that they are monic. Therefore $f(x,\beta)$ and $g(x,\beta)$ are coprime with probability $1 - 1/q$. Since we have q choices for β we obtain

$$E[X] = \sum_{\beta \in \mathbb{F}_q} \mathrm{Prob}[\gcd(A(x,\beta), B(x,\beta)) \neq 1] = q(1 - (1 - \frac{1}{q})) = 1.$$

Proof of claim. Since $B = \{1, y-\beta, (y-\beta)^2, \ldots, (y-\beta)^d\}$ is a basis for polynomials of degree d we can write each $C(y) = \sum_{i=0}^{d} c_i y^i$ as $C(y) = u_0 + \sum_{i=1}^{d} u_i(y-\beta)^i$ for a unique choice of $u_0, u_1, \ldots, u_d \in \mathbb{F}_q$. Since $C(\beta) = u_0$ it follows that all q^d choices for $u_1, \ldots, u_d$ result in $C(\beta) = u_0$ hence C is balanced. □

That $E[X] = 1$ was a surprise to us. We thought $E[X]$ would have a logarithmic dependence on $\deg f$ and $\deg g$. In light of Theorem 3.3, when picking $p > \deg_y(K_r(H))$ we will ignore the unlucky evaluation points. If the algorithm encounters unlucky evaluations, then we restart the algorithm with a larger prime.

3.3 The first example revisit

In our GCD algorithm with a Kronecker substitution we only need to interpolate one variable, say y in our case. The smooth primes of the form $2^k \cdot s + 1$ are easy to find. Therefore it somehow simplifies our algorithm. In this section we redo the first example in Section 2.4 with a Kronecker substitution.

Example 3.6. *Consider the GCD problem $A = \bar{A}G$ and $B = \bar{B}G$, where*

$$G = (92y^2 - 513z)x^2 + (212y^2 + yz^2 + 125z)x + (251y^2z^2 - 43z^3 + 5y^2 + 318),$$

$$\bar{A} = y^2x + z \quad \textit{and} \quad \bar{B} = y^3x^2 + z.$$

Let x be the main variable and $\Gamma = \gcd(LC(A), LC(B)) = y^2(92y^2 - 513z)$. We first need to determine the degree of the target GCD in y to set up the Kronecker substitution. We pick a random evaluation point for x and z and use DegreeBound algorithm in Figure 2.3 to obtain $\deg_y G \leq 2$. We have to consider the contribution of degree of Γ in y which is $\deg_y \Gamma = 4$ because G could be monic. Hence the degree of the scaled GCD in y is at most $4 + 2 = 6$. But $\deg_y A = 4, \deg_y B = 5$, the degree of the scaled GCD in y cannot be larger than $\min\{\deg_y A, \deg_y B\} = 4$, and therefore we pick $r = \deg_y A + 1 = 5$ for the Kronecker substitution for the first try.

Since $\deg_y K_r(A) = \deg_y K_r(B) = 20$, we first use the prime $p = 2^6 \cdot 3 + 1 = 193$ which is larger than 20. We pick $\omega = 5$ as a generator of $\mathbb{Z}^*_{193}$.

The purpose of this example is to demonstrate the role of Kronecker substitution in this new GCD algorithm. We also note that $LC(\gcd(\bar{A}, \bar{b}) = y^2$ is a monomial and therefore the sparsity is preserved after the scaling. We skip the GCD iteration steps and assume we know $t = 4$ and compute 8 univariate GCD images in a row with the random shift $s = 5$.

$$\begin{array}{lrcrcr}
K_r(\Gamma)(\omega^{s+0})\gcd(K_r(A)(x,\omega^{s+0}), K_r(B)(x,\omega^{s+0})) = & 167 & x^2 & +106 & x & +147 \\
K_r(\Gamma)(\omega^{s+1})\gcd(K_r(A)(x,\omega^{s+1}), K_r(B)(x,\omega^{s+1})) = & 137 & x^2 & +104 & x & +132 \\
K_r(\Gamma)(\omega^{s+2})\gcd(K_r(A)(x,\omega^{s+2}), K_r(B)(x,\omega^{s+2})) = & 74 & x^2 & +154 & x & +61 \\
K_r(\Gamma)(\omega^{s+3})\gcd(K_r(A)(x,\omega^{s+3}), K_r(B)(x,\omega^{s+3})) = & 80 & x^2 & +122 & x & +125 \\
K_r(\Gamma)(\omega^{s+4})\gcd(K_r(A)(x,\omega^{s+4}), K_r(B)(x,\omega^{s+4})) = & 189 & x^2 & +79 & x & +135 \\
K_r(\Gamma)(\omega^{s+5})\gcd(K_r(A)(x,\omega^{s+5}), K_r(B)(x,\omega^{s+5})) = & 110 & x^2 & +35 & x & +93 \\
K_r(\Gamma)(\omega^{s+6})\gcd(K_r(A)(x,\omega^{s+6}), K_r(B)(x,\omega^{s+6})) = & 55 & x^2 & +107 & x & +90 \\
K_r(\Gamma)(\omega^{s+7})\gcd(K_r(A)(x,\omega^{s+7}), K_r(B)(x,\omega^{s+7})) = & 80 & x^2 & +160 & x & +38
\end{array}$$

We mention that the coefficient of x^2 is the scaling factor Γ which is known. We extract the coefficients of x^1 and x^0 from univariate GCD images to form two sequences

$$[106, 104, 154, 122, 79, 35, 107, 160] \quad \textit{and} \quad [147, 132, 61, 125, 135, 93, 90, 38]$$

on which we apply the Berlekamp-Massey algorithm and obtain the feedback polynomials

$$c_1(z) = 150z^3 + 80z^2 + 60z + 1 \quad \textit{and} \quad c_2(z) = 57z^4 + 43z^3 + 150z^2 + 14z + 1.$$

The reciprocals of both polynomials are $\Lambda_1 = z^3 + 60z^2 + 80z + 150$ and $\Lambda_2 = z^4 + 14z^3 + 150z^2 + 43z + 57$. The root finding algorithm returns roots $\{153, 46, 127\}$ and $\{25, 46, 52, 56\}$ and the corresponding monomials are $\{y^7, y^4, y^{13}\}$ and $\{y^2, y^4, y^{17}, y^{14}\}$. By the shifted Vandermonde system solving algorithm with $s = 5$ we obtain the first modular scaled GCD image

$$H_1 = (66y^7 + 92y^4)x^2 + (y^{13} + 125y^7 + 19y^4)x + 150y^{17} + 58y^{14} + 5y^4 + 125y^2.$$

Then we assume the complete support $supp(H_1)$ is obtained and set a form by $supp(H_1)$. By Zippel's sparse interpolation with a new random prime, say $p = 101$, we obtain

$$H_2 = (93y^7 + 92y^4)x^2 + (y^{13} + 24y^7 + 10y^4)x + 58y^{17} + 49y^{14} + 5y^4 + 15y^2.$$

For an example of using Zippel's sparse interpolation to compute GCD, see Example 2.4. The Chinese remaindering with symmetric representation modulo $193 \cdot 101$ on inputs H_1 and H_2 gives us

$$\hat{H}_1 = (-513y^7 + 92y^4)x^2 + (y^{13} + 125y^7 + 212y^4)x - 43y^{17} + 251y^{14} + 5y^4 + 318y^2.$$

We need one more image to check whether $\hat{H}_1$ is stable. By Zippel's sparse interpolation again with the new prime 103, we have

$$H_3 = (2y^7 + 92y^4)x^2 + (y^{13} + 22y^7 + 6y^4)x + 60y^{17} + 45y^{14} + 5y^4 + 9y^2.$$

By applying the Chinese remaindering to $\hat{H}_1$ and H_3 we have

$$\hat{H}_2 = (-513y^7 + 92y^4)x^2 + (y^{13} + 125y^7 + 212y^4)x - 43y^{17} + 251y^{14} + 5y^4 + 318y^2.$$

Since $\hat{H}_1 = \hat{H}_2$, $\hat{H}_2$ is likely to be the correct result. We apply the inverse Kronecker substitution to $\hat{H}_2$ and obtain

$$K_r^{-1}(\hat{H}_2) = (92y^4 - 513y^2z)x^2 + (y^3z^2 + 212y^4 + 125y^2z)x + 251y^4z^2 - 43y^2z^3 + 5y^4 + 318y^2.$$

Since y^2 is the content of $K_r^{-1}(\hat{H}_2)$ with respect to x, we divide it out and get

$$\hat{G} = (92y^2 - 513z)x^2 + (yz^2 + 212y^2 + 125z)x + 251y^2z^2 - 43z^3 + 5y^2 + 318.$$

We test $\hat{G}|A$ and $\hat{G}|B$ and conclude that $\hat{G} = G$.

Remark 3.1. So far we use random evaluation points to construct linear systems in Zippel's sparse interpolation to solve the coefficients of the assumed form. Classically each linear system costs $O(t^3)$ operations to get a solution. This could be a bottleneck of our GCD algorithm if t is large. If we use the evaluation sequence described in Ben-Or/Tiwari interpolation with discrete logarithm method, that is $\{\omega^i \mid i = 0, 1, 2, , \dots\}$, then each linear system is a Vandermonde system which can be solved efficiently by Zippel's method with $O(t^2)$ operations. See Section 2.7. With fast interpolating, the cost can be accelerated to $O(M(t) \log t)$. See Table 2.2. This will be discussed in our faster GCD algorithm.

Chapter 4

Simplified Algorithm

In this chapter we assemble a "Simplified Algorithm" which is a Las-Vegas GCD algorithm. This algorithm consists of two parts: the main routine MGCD and the subroutine PGCD. PGCD computes the GCD modulo a prime and MGCD calls PGCD several times to obtain enough images to reconstruct the coefficients of the target polynomial H by Chinese Remaindering. In this section, we assume that we are given a term bound τ on the number of terms in the coefficients of target polynomial H with respect to x_0, that is $\tau \geq \#h_i(x_1, x_2, \ldots, x_n)$. This implies that the Berlekamp-Massey algorithm will always terminate. In this simplified algorithm, we use $\Gamma = \gcd(LC(A), LC(B))$ rather than $LC(A)$ or $LC(B)$ to scale the univariate images because the Berlekamp-Massey algorithm always terminates and therefore the algorithm will not run into an infinite loop. We will also choose a Kronecker substitution that is a priori not bad and not unlucky. These assumptions will enable us to choose a prime p so that PGCD computes G with high probability.

4.1 Bad and unlucky Kronecker substitutions

Lemma 4.1. *Let $K_r : \mathbb{Z}[x_0, x_1, \ldots, x_n] \to \mathbb{Z}[x, y]$ be the Kronecker substitution $K_r(f) := f(x, y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{n-1}})$. If $f \neq 0$ and $r_i > \deg_{x_i}(f)$ for $1 \leq i \leq n-1$ then $K_r(f)$ sends monomials in f to unique monomials and therefore K_r is one-to-one and $K_r(f) \neq 0$.*

Proof. Suppose two monomials $x_0^{d_0} x_1^{d_1} \cdots x_n^{d_n}$ and $x_0^{e_0} e_1^{e_1} \cdots x_n^{e_n}$ in f are mapped to the same monomial in $\mathbb{Z}[x, y]$ so that

$$x^{d_0} y^{d_1} y^{r_1 d_2} \cdots y^{r_1 r_2 \cdots r_{n-1} d_n} = x^{e_0} y^{e_1} y^{r_1 e_2} \cdots y^{r_1 r_2 \cdots r_{n-1} e_n}$$

Clearly $d_0 = e_0$ and

$$d_1 + r_1 d_2 + \cdots + r_1 r_2 \cdots r_{n-1} d_n = e_1 + r_1 e_2 + \cdots + r_1 r_2 \cdots r_{n-1} e_n \tag{4.1}$$

Reducing (4.1) modulo r_1 we have $d_1 \equiv e_1 \pmod{r_1}$. Now $r_1 > \deg_{x_1} f$ implies $r_1 > d_1$ and $r_1 > e_1$ implies $d_1 = e_1$. Subtracting $d_1 = e_1$ from this equation and dividing through by r_1

we have

$$d_2 + r_2 d_3 + \ldots r_2 r_3 \cdots r_{n-1} d_n = e_2 + r_2 e_3 + \ldots r_2 r_3 \cdots r_{n-1} e_n$$

Repeating the argument we obtain $d_i = e_i$ for $1 \leq i \leq n$. □

In our case, we are considering the polynomials $A, B \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$ with $\deg_{x_0} A > 0$ and $\deg_{x_0} B > 0$. Let $G = \gcd(A, B)$ and $\bar{A} = A/G$ and $\bar{B} = B/G$ be cofactors of A and B and let $LC(A)$ and $LC(B)$ the the leading coefficients of A and B with respect to x_0. Lemma 4.1 implies that if we pick $r_i > \max(\deg_{x_i} LC(A), \deg_{x_i} LC(B))$ then $K_r(LC(A)) \neq 0$ and $K_r(LC(B)) \neq 0$ thus K_r is not bad. Let $R = \mathrm{res}_{x_0}(\bar{A}, \bar{B})$. By Lemma 2.4 (iii), we have

$$\deg_{x_i} R \leq \deg x_0 \bar{B} \deg_{x_i} \bar{A} + \deg x_0 \bar{A} \deg_{x_i} \bar{B}.$$

Since $\deg_{x_i} \bar{A} \leq \deg_{x_i} A$ and $\deg_{x_i} \bar{B} \leq \deg_{x_i} B$ for $0 \leq i \leq n$ we have

$$\deg_{x_i} R \leq \deg_{x_0} B \deg_{x_i} A + \deg_{x_0} A \deg_{x_i} B.$$

So if we pick $r_i = (\deg_{x_0} B \deg_{x_i} A + \deg_{x_0} A \deg_{x_i} B) + 1$, then K_r is not unlucky by Lemma 4.1. The assumption that $\deg_{x_0} A > 0$ and $\deg_{x_0} B > 0$ gives

$$\deg_{x_0} B \deg_{x_i} A + \deg_{x_0} A \deg_{x_i} B \geq \max\{\deg_{x_i} LC(A), \deg_{x_i} LC(B))\}$$

hence the Kronecker substitution K_r with the sequence

$$[r_i = (\deg_{x_i} A \deg_{x_0} B + \deg_{x_i} B \deg_{x_0} A) + 1 : 1 \leq i \leq n] \tag{4.2}$$

is not bad and not unlucky.

Remark 4.1. Since $\deg_{x_0} A > 0$ and $\deg_{x_0} B > 0$, Equation (4.2) implies $r_i > \deg_{x_i} A$ and $r_i > \deg_{x_i} B$. Therefore the map K_r is invertible for A and B and $\#K_r(A) = \#A$ and $\#K_r(B) = \#B$.

4.2 Bad and unlucky evaluations

In this section, the Kronecker substitution K_r is assumed to be good. We also assume that the prime p is good.

Proposition 4.1. *Let* $r_i = (\deg_{x_i} A \deg_{x_0} B + \deg_{x_i} B \deg_{x_0} A) + 1$ *for* $1 \leq i \leq n$, *and* $d = \max\{\max\{\deg_{x_i} A, \deg_{x_i} B\}_{0 \leq i \leq n}\}$. *Then we have*

(1) $\deg_y K_r(A) \leq (2d^2+1)^n$ *and* $\deg_y K_r(B) \leq (2d^2+1)^n$,
(2) $\deg_y LC(K_r(A))(y) \leq (2d^2+1)^n$ *and* $\deg_y LC(K_r(B))(y) \leq (2d^2+1)^n$,
(3) $\deg_y K_r(H) \leq (2d^2+1)^n$, *and*
(4) $\deg_y K_r(R) \leq 2d(2d^2+1)^n$, *where* $K_r(R) = \mathrm{res}_x(K_r(\bar{A}), K_r(\bar{B}))$.

Proof. For (1), after the Kronecker substitution, the exponent of $y \le e_1 + e_2(2d^2+1) + \cdots + e_n(2d^2+1)^{n-1}$, where e_i is the exponent of x_i and $e_i \le d$ for all i. So $\deg_y K_r(A)$ and $\deg_y K_r(B)$ are bounded by

$$\begin{aligned} d + d(2d^2+1) + \cdots + d(2d^2+1)^{n-1} &= d(1 + (2d^2+1) + \cdots + (2d^2+1)^{n-1}) \\ &= d(1 + \frac{(2d^2+1)^n - (2d^2+1)}{(2d^2+1) - 1}) \\ &= \frac{2d^3}{2d^2} + \frac{d(2d^2+1)^n - d(2d^2+1)}{2d^2}) \\ &= \frac{d(2d^2+1)^n - d}{2d^2} \\ &< (2d^2+1)^n. \end{aligned}$$

Property (2) follows from (1). For (3), recall that $\deg_y K_r(H) = \deg_y K_r(\Delta G)$. Since $\Delta = \gcd(\mathrm{LC}(\bar{A}), LC(\bar{B}))$, we have

$$\begin{aligned} \deg_y K_r(\Delta G) &= \deg_y K_r(\Delta) + \deg_y K_r(G) \\ &\le \min(\deg_y K_r(LC(\bar{A})), \deg_y K_r(LC(\bar{B}))) + \deg_r K_r(G) \\ &\le \min(\deg_y K_r(\bar{A}), \deg_y K_r(\bar{B})) + \deg_r K_r(G) \\ &= \min(\deg_y K_r(A), \deg_y K_r(B)) \le (2d^2+1)^n. \end{aligned}$$

For (4),

$$\deg_y K_r(R) \le \deg_y K_r(\bar{A}) \deg_x K_r(\bar{B}) + \deg_y K_r(\bar{B}) \deg_x K_r(\bar{A}),$$

where $\deg_x K_r(\bar{A}) = \deg_{x_0} \bar{A} \le \deg_{x_0} A \le d$, $\deg_x K_r(\bar{B}) = \deg_{x_0} \bar{B} \le \deg_{x_0} B, \le d$, and $\deg_y K_r(\bar{A}) \le \deg_y K_r(A)$ and $\deg_y K_r(\bar{B}) \le \deg_y K_r(B)$. So we have

$$\deg_y K_r(R) < d(2d^2+1)^n + d(2d^2+1)^n = 2d(2d^2+1)^n.$$

□

By Proposition 4.1(1), a prime $p > (2d^2+1)^n$ is sufficient to recover the exponents for the Kronecker substitution. With the assumption that p is not bad and not unlucky, we have the following two lemmas.

Lemma 4.2. *Let p be a good prime. If α is chosen uniformly at random from $[0, p-1]$, then*

$$Prob(\alpha \text{ is bad}) = Prob(LC(K_r(A))(\alpha) LC(K_r(B))(\alpha) = 0) \tag{4.3}$$

$$\le \frac{\deg LC(K_r(A))(y) + \deg LC(K_r(B))(y)}{p} < \frac{2(2d^2+1)^n}{p}. \tag{4.4}$$

Lemma 4.3. *Let p be a good prime. If α is chosen uniformly at random from $[0, p-1]$, then*

$$Prob(\alpha \text{ is bad or unlucky}) < \frac{2(2d^2+1)^n + 2d(2d^2+1)^n}{p}.$$

Proof. This result follows from Proposition 4.1(4), Lemma 4.2 and Lemma 2.7. □

The probability that our algorithm does not encounter a bad or unlucky evaluation can be estimated as follows. Let U denote the bound of the number of bad and unlucky evaluation points and $\tau \geq max_i\{\#h_i\}$. We need 2τ good consecutive evaluation points (a segment of length 2τ in the sequence $(1, .., p-1)$) to compute the feedback polynomial for h_i. Suppose α^k is a bad or unlucky evaluation point where $s \leq k < 2\tau$ for any positive integer $s \in (0, p-1]$. Then every segment of length 2τ starting at α^i where $k - 2\tau + 1 \leq i \leq k$ includes the point α^k so that our algorithm fails to determine the correct feedback polynomial due to bad or unlucky evaluation. The union of all segments including α^k has length $4\tau - 1$. We can not use every segment of length 2τ from $k - 2\tau + 1$ to $k + 2\tau - 1$ to construct the correct feedback polynomial. The worst case occurs when for all bad and unlucky evaluation points, their corresponding segments of length $4\tau - 1$ do not overlap. Since there are at most U of them, we can not determine the correct feedback polynomials for at most $U(4\tau - 1)$ points. Note, this does not mean that all those points are bad or unlucky, there is only one bad or unlucky point in each segment of length 2τ. U is bounded by $2(2d^2+1)^n + 2d(2d^2+1)^n = (2d+2)(2d^2+1)^n$.

Lemma 4.4. *Suppose p is good.*

$$Prob(2\tau \text{ evaluation points } fail \text{ to determine the } feedback \text{ polynomial})$$
$$\leq \frac{4\tau U - U}{p-1} < \frac{4\tau U}{p-1} \leq \frac{4\tau(2d+2)(2d^2+1)^n}{p-1} \leq \frac{1}{X},$$

for some positive number X. So if we choose a prime $p > 4X\tau(2d+2)(2d^2+1)^n$, then the probability that PGCD fails is at most $\frac{1}{X}$.

Remark 4.2. The choice of p in previous lemma implies $p > (2d^2+1)^n \geq \deg_y(K_r(H))$. So we can recover the exponents of y in H.

4.3 Bad and unlucky primes

Our goal here is to construct a set S of smooth primes, with $|S|$ large enough so if we choose a prime $p \in S$ uniformly at random, the probability that p is good is at least $\frac{1}{2}$.

A bad prime must divide $\|LC(K_r(A))\|$ or $\|LC(K_r(B))\|$ and an unlucky prime must divide $\|K_r(R)\|$. Recall that in section 2.1,

$$M = \|LC(K_r(A))\|\|LC(K_r(B))\|\|Kr(R)\|.$$

We want to construct a set $S = \{p_1, p_2, \ldots, p_N\}$ of N smooth primes with each $p_i > 4\tau(2d+4)(2d^2+1)^n X$. If $p > 4\tau(2d+4)(2d^2+1)^n X$, then the probability that our algorithm fails to determine the feedback polynomial is $< \frac{1}{X}$. The size N of S can be estimated as follows. If

$$N = Y\lceil \log_{4X\tau(2d+4)(2d^2+1)^n} M \rceil > Y \log_{pmin} M,$$

where a bound for M is given by Theorem 3.2 (ii), $pmin = \min_{p_i \in S} p_i$ and $Y > 0$, then

$$\text{Prob}(\ p \textit{ is bad or unlucky}) \leq \frac{\log_{pmin} M}{N} < \frac{1}{Y}.$$

We construct a set S which consists of N y-smooth primes where $y \in \mathbb{N}$ so that $\min_{p_i \in S} p_i > 4\tau X(2d+4)(2d^2+1)^n$ which is the constraint for the bad or unlucky evaluation case. The size of y affects the efficiency of the discrete logarithm computation. We conclude with the following result.

Theorem 4.1. *Let S be constructed as just described. Let p be chosen uniformly at random from S, s be chosen uniformly at random from $0 < s \leq p-1$ and α_p be a random generator of $\mathbb{Z}_p^*$. Let $E = \{\alpha_p^{s+j} : 0 \leq j < 2\tau\}$ be 2τ consecutive evaluation points. For given $X > 0$ and $Y > 0$, we have*

$$\textit{Prob}(p \textit{ is good and } E \textit{ are all good}) > (1 - \frac{1}{X})(1 - \frac{1}{Y}).$$

4.4 Algorithm

Let $S = \{p_1, p_2, \ldots, p_N\}$ be the set of N primes constructed in the previous section. We've split our GCD algorithm into two subroutines, subroutine MGCD and PGCD. The main routine MGCD chooses a Kronecker substitution K_r and then chooses a prime p from S uniformly at random and calls PGCD to compute $K_r(H) \bmod p$.

Algorithm MGCD is a Las Vegas algorithm. Let $H = \sum_{i=0}^{d_0} h_i x_0^i$. The choice of S means that algorithm PGCD will compute $K_r(H) \mod p$ with probability at least $(1-\frac{1}{X})(1-\frac{1}{Y})$. By taking $X = 4$ and $Y = 4$ this probability is at least $\frac{1}{2}$. The design of MGCD means that even with probability $\frac{1}{2}$, the expected number of calls to algorithm PGCD is linear in the minimum number of primes needed to recover H using Chinese remaindering, that is, we do not need to make the probability that algorithm PGCD computes $H \mod p$ high for algorithm MGCD to be efficient.

Algorithm MGCD(A, B, τ)

Inputs $A, B \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$ and a term bound τ satisfying $n > 0$, A and B are primitive in x_0, $\deg_{x_0} A > 0$, $\deg_{x_0} B \geq 0$ and $\tau \geq \max \#h_i$.

Output $G = \gcd(A, B)$.

1 Compute $\Gamma = \gcd(LC(A), LC(B))$ in $\mathbb{Z}[x_1, \ldots, x_n]$.

2 Set $r_i = 1 + (\deg_{x_i} A \deg_{x_0} B + \deg_{x_i} B \deg_{x_0} A)$ for $1 \leq i < n$.

3 Let $Y = (y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{n-1}})$.
Set $K_r(A) = A(x, Y)$, $K_r(B) = B(x, Y)$ and $K_r(\Gamma) = \Gamma(Y)$.

4 Construct the set S of smooth primes according to Theorem 4.1 with $X = 4$ and $Y = 4$.

5 Set $\widehat{H} = 0, M = 1, d_0 = \min(\deg_{x_0} A, \deg_{x_0} B)$.

LOOP: // Invariant: $d_0 \geq \deg_{x_0} H = \deg_{x_0} G$.

6 Call PGCD($K_r(A)$, $K_r(B)$, $K_r(\Gamma)$, S, τ, M).
If PGCD outputs FAIL then **goto** LOOP.
Let p and $\widehat{H}p = \sum_{i=0}^{dx} \widehat{h}_i(y) x^i$ be the output of PGCD.

7 If $dx > d_0$ then either p is unlucky or all evaluation points were unlucky so **goto** LOOP.

8 If $dx < d_0$ then either this is the first image or all previous images in $\widehat{H}$ were unlucky so set $d_0 = dx$, $\widehat{H} = Hp$, $M = p$ and **goto** LOOP.

Chinese-Remaindering

9 Set $Hold = \widehat{H}$. Solve $\{\widehat{H} \equiv Hold \mod M$ and $\widehat{H} \equiv \widehat{H}p \mod p\}$ for $\widehat{H}$. Set $M = M \times p$. If $\widehat{H} \neq Hold$ then **goto** LOOP.

Termination.

10 Set $\widetilde{H} = K_r^{-1} \widehat{H}(x, y)$ and let $\widetilde{H} = \sum_{i=0}^{d_0} \widetilde{C}_i x_0^i$ where $\widetilde{C}_i \in \mathbb{Z}[x_1, x_2, \ldots, x_n]$.

11 Set $\widehat{G} = \widetilde{H} / \gcd(\widetilde{C}_0, \widetilde{C}_1, \ldots, \widetilde{C}_{d_0})$ ($\widehat{G}$ is the primitive part of $\widetilde{H}$).

12 If $\widehat{G} | A$ and $\widehat{G} | B$ then **output** $\widehat{G}$.

13 **goto** LOOP.

Algorithm PGCD($K(A)$, $K(B)$, $K(\Gamma)$, S, τ, M)

Inputs $K(A), K(B) \in \mathbb{Z}[x, y]$, $K(\Gamma) \in \mathbb{Z}[y]$, S a set of smooth primes, a term bound $\tau \geq \max \#h_i$ and M a positive integer.

Output With probability $\geq \frac{1}{2}$ a prime p and polynomial $Hp \in \mathbb{Z}_p[x, y]$ satisfying $Hp = K(H) \bmod p$ and p does not divide M, otherwise FAIL.

1 Pick a prime p uniformly at random from S that is not bad and does not divide M.

2 Pick a shift $s \in \mathbb{Z}_p^*$ uniformly at random and any generator α for $\mathbb{Z}_p^*$.

Compute-and-scale-images:

3 For j from 0 to $2\tau - 1$ do

4 Compute $a_j = K(A)(x, \alpha^{s+j}) \bmod p$ and $b_j = K(B)(x, \alpha^{s+j}) \bmod p$.

5 If $\deg_x a_j < \deg_x K(A)$ or $\deg_x b_j < \deg_x K(B)$ then **output** FAIL (α^{s+j} is a bad evaluation point.)

6 Compute $g_j = \gcd(a_j, b_j) \in \mathbb{Z}_p[x]$ using the Euclidean algorithm and set $g_j = K(\Gamma)(\alpha^{s+j}) \times g_j \bmod p$.

End for loop.

7 Set $d_0 = \deg g_0(x)$. If $\deg g_j(x) \neq d_0$ for any $1 \leq j \leq 2\tau - 1$ **output** FAIL (unlucky evaluations).

Interpolate-coefficients:

8 For $i = 0$ to $d_0 - 1$ do

9 Run the Berlekamp-Massey algorithm on the coefficients of x^i in the images $g_0, g_1, \ldots, g_{2\tau-1}$ to obtain the feedback polynomial $c_i(z)$ and set $\tau_i = \deg c_i(z)$.

10 Compute the reciprocal of $c_i(z)$ to get $\Lambda_i(z)$ and then compute the roots m_j of each $\Lambda_i(z)$ in $\mathbb{Z}_p$. If the number of distinct roots of $\Lambda_i(z)$ is not equal τ_i then **output** FAIL (the feedback polynomial is wrong due to undetected unlucky evaluations.)

11 Set $e_k = \log_\alpha m_k$ for $1 \leq k \leq \tau_i$ and let $\sigma_i = \{y^{e_1}, y^{e_2}, \ldots y^{e_{\tau_i}}\}$.

12 Solve the τ_i by τ_i shifted transposed Vandermonde system

$$\left\{ \sum_{k=1}^{\tau_i} (\alpha^{s+j})^{e_k} u_k = \text{coefficient of } x^i \text{ in } g_j(x) \quad \textbf{for } 0 \leq j < \tau_i \right\}$$

modulo p for u and set $h_i(y) = \sum_{k=1}^{\tau_i} u_k y^{e_k}$. Note: $(\alpha^{s+j})^{e_k} = m_k^{s+j}$

End for loop.

13 Set $Hp := K(\Gamma)x^{d_0} + \sum_{i=0}^{d_0-1} h_i(y)x^i$ and **output** (p, Hp).

Theorem 4.2. *Let $pmin = \min S$ and let $N = \log_{pmin} \|2H\|$. So N primes in S are sufficient to recover the integer coefficients of H using Chinese remaindering. Let Z be the number of calls that Algorithm MGCD makes to Algorithm PGCD. Then $E[Z] \leq 2(N+1)$.*

Proof. Because the Kronecker substitution K_r is not bad, and the primes p used in PGCD are not bad and the evaluations points $\{\alpha^{s+j} : 0 \leq j \leq 2\tau - 1\}$ used in PGCD are not bad, in Step 6 of Algorithm PGCD, $\deg g_j(x) \geq \deg_{x_0} G$ by 2.6. Therefore $d_0 \geq \deg_{x_0} H =$

$\deg_{x_0} G$ throughout Algorithm MGCD and $\deg_x \widehat{H} = \deg_{x_0} \widehat{G} \geq \deg_{x_0} G$. Since A and B are primitive in x_0, if $\widehat{G}|A$ and $\widehat{G}|B$ then it follows that $\widehat{G} = G$, so if algorithm MGCD terminates, it outputs G.

To prove termination observe that Algorithm MGCD proceeds in two phases. In the first phase MGCD loops while $d_0 > \deg_{x_0} H$. In this phase no useful work is accomplished. Observe that the loops in PGCD are of fixed length 2τ and d_0+1 so PGCD always terminates and algorithm MGCD tries another prime. Because at least 3/4 of the primes in S are good, and, for each prime, at least 3/4 of the possible evaluation point sequences are good, eventually algorithm PGCD will choose a good prime and a good evaluation point sequence after which $d_0 = \deg_{x_0} H$.

In the second phase MGCD loops using images Hp with $\deg_x Hp = d_0$ to construct $\widehat{H}$. Because the images $g_j(x)$ used satisfy $\deg_x g_j(x) = d_0 = \deg_{x_0} H$ and we scale them with $\Gamma(\alpha^{s+j})$, PGCD interpolates $Hp = H \mod p$ thus we have modular images of H. Eventually $\widehat{H} = H$ and the algorithm terminates.

Because the probability that the prime chosen from S is good is at least 3/4 and the evaluation points α^{s+j} are all good is at least 3/4, the probability that PGCD outputs a good image of H is at least 1/2. Since N images of H are sufficient to recover $2H$ and we meed one more to stabilize, $E[Z] \leq 2(N+1)$ as claimed. □

Remark 4.3. Note, we do not check for termination after each prime because computing the primitive part of $\widetilde{H}$ or doing the trial divisions $\widehat{G}|A$ and $\widehat{G}|B$ in step 12 could be more expensive than algorithm PGCD. Instead algorithm MGCD waits until the Chinese remaindering stabilizes in Step 9 before proceeding to test for termination.

Chapter 5

Faster Algorithm

In this chapter, we consider the practical improvement of our simplified algorithm. The new algorithm consists of three algorithms: MGCD1, PGCD1 and SGCD1. In practice, because the term bound $\tau \geq t$ is usually unknown, the method discussed in Section 2.5 will be used. The Kronecker substitution used in the simplified algorithm is always not bad and not unlucky which leads to use larger primes. In order to handle GCD problems as large as possible with 63 bit primes or 127 bit primes as we have implemented, we use smaller Kronecker substitution which guarantees the invertibility of $K_r(H)$ but may encounter unlucky Kronecker substitutions. We make three improvements, one necessary, the other two efficiency improvements. Unfortunately, each improvement leads to a major complication which we solve probabilistically.

5.1 Term Bounds

Recall that $H = \Delta G = \sum_{i=0}^{dG} h_i(x_1, \ldots, x_n)x_0^i$. Algorithms MGCD and PGCD assume a term bound τ on $\#h_i(y)$. In practice, good term bounds are usually not available. For the GCD problem, one cannot even assume that $\#G \leq \min(\#A, \#B)$. So we must modify the algorithm to compute $t_i = \#h_i(y)$.

We have provided details to determine t_i in chapter 2. To be precise, we will loop calling the Berlekamp-Massey algorithm after $2, 4, 6, 8, \ldots$, evaluation points and wait until we get two consecutive zero discrepancies. This means $c(z)$ is correct with high probability when p is sufficiently large, see Theorem 2.6. This loop will terminate. However, the support σ_i of $K_r(h_i)$ computed in step 11 of PGCD1 may still be wrong due to early stabilization of the feedback polynomial. Note that even if the feedback polynomial stabilizes at degree d, the algorithm continues to loop unless the reciprocal of the feedback polynomial has exactly d distinct roots. We consider an example.

Example 5.1. *Consider the following GCD problem in $\mathbb{Z}[x, y]$. Let p and q be prime and let*

$$G = x + py + qy^2 + py^4,\ \bar{A} = 1,\ \bar{B} = 1.$$

Suppose MGCD chooses p first and suppose PGCD returns $x + qy^2 \mod p$ so that $\sigma_0 = \{y^2\}$. Suppose MGCD chooses q next and suppose $c_0(z)$ stabilizes too early and $\sigma_0 = \{y^3\}$ which is wrong. This could also be due to missing terms, for example, if $G = x + pqy + qy^2 + py^3$. If we combine these two images modulo p and q using Chinese remaindering to obtain $\widehat{H}$ of the form $x + \cdot y^2 + \cdot y^3$ we have a bad image in $\widehat{H}$ and we need somehow to detect it. Once detected, we do not want to restart the entire algorithm because we might be throwing away a lot of good images in $\widehat{H}$. Our solution in Steps 8–11 of algorithm MGCD1 is probabilistic.

5.2 Using smaller primes

Another consideration is to reduce r_i in the Kronecker substitution hence reduce the size of the primes. The Kronecker substitution is required to be invertible for H but may be unlucky. We have implemented our GCD algorithm for 63 bit primes and 127 bit primes. With smaller r_i, we can solve GCD problems of higher degree or more variables with 63 bit or 127 bit primes. By choosing a Kronecker substitution that is a priori good, and requiring that the 2τ evaluation points are good, the primes in S constructed for the simplified algorithm must be greater than $4\tau(2d+2)(2d^2+1)^n$ where d bounds the degree of A and B in all variables. Instead if we choose $r_i > \deg_{x_i} H$ then we will still be able to recover H from $K_r(H)$ by inverse map of the Kronecker substitution.

Since $\deg_{x_i} H \leq \min(\deg_{x_i} A, \deg_{x_i} B) \leq d$, using $r_i = d + 1$ we replace the factor $(2d^2+1)^n$ with $(d+1)^n$. The unlucky K_r can be detected when $\deg g_j(x) > d_0$ in step 6 of PGCD1 by computing $d_0 = \mathit{DegreeBound}(A, B, 0)$ periodically so that eventually we obtain $d_0 = \deg_{x_0} G$. Once detected we increase r_i by 1 to try a larger Kronecker substitution.

Recall that p is an unlucky prime if $p|R$ where $R = \mathrm{res}_{x_0}(\bar{A}, \bar{B})$. Because the inputs A and B are primitive in x_0 it follows that the integer coefficients of A and B and hence also $\bar{A}$ and $\bar{B}$ are relatively prime. Therefore, the integer coefficients of R are also likely relatively prime, and if not, the common factor is likely small e.g., 2. Thus the expected number of unlucky primes is very close to 0. Theorem 3.3 showed that the expected number of unlucky evaluations is also very low and hence instead of using $p > 4\tau(2d+2)(d+1)^n$ we first try a prime $p > 4(d+1)^n$. This reduces the length of the primes from size $4\tau(2d+2)(2d^2+1)^n$ to size $4(d+1)^n$ for most inputs by at least a factor of 2, equivalently, it allows us to handle twice as many variables or to square the degree of y for a fixed prime size. Once we encounter bad or unlucky evaluation points we increase the length of p.

Example 5.2. *For our benchmark problem where $n = 8$, $d = 20$ and $\tau = 1000$ we have $\log_2[4\tau(2d+2)(2d^2+1)^n] = 94.5$ bits which precludes our using 63 bit primes. On the other hand $\log_2[4(d+1)^n] = 37.1$ bits, meaning a 63 bit prime is more than sufficient.*

5.3 Using fewer evaluation points

Let $K_r(h_i) = \sum_{j=1}^{t_i} C_{ij} y^{e_{ij}}$ for some coefficients $C_{ij} \in \mathbb{Z}$ so that $\text{Supp}(K_r(h_i)) = \{y^{e_{ij}} : 1 \le j \le t_i\}$. Because of the size of the primes chosen by algorithm MGCD, it is likely that the first good image Hp computed by PGCD has the entire support of $K_r(H)$, that is, $\text{Supp}(\widehat{h}_i) = \text{Supp}(K_r(h_i))$. Assuming this to be so, we can compute the next image of $K_r(H)$ modulo p using only t evaluations instead of $2t+O(1)$ as follows. We choose a prime p and compute $g_j(x)$ for $0 \le j < t$ as before in PGCD. Suppose these t images are all good, α is a generator of $\mathbb{Z}_p^*$ and s is the shift value chosen uniformly at random from $\mathbb{Z}_p$. One may solve the the t_i by t_i shifted transposed Vandermonde systems

$$\left\{ \sum_{j=1}^{t_i} (\alpha^{s+j})^{e_{ij}} u_{ij} = \text{coefficient of } x^i \text{ in } g_j(x) \ \text{ for } 0 \le j \le \tau_i \right\}$$

for the unknown coefficients u_{ij} and obtain $Hp = \sum_{i=0}^{d_0} \sum_{j=0}^{t_i} u_{ij} y^{e_{ij}}$. This method, named as SGCD1, chooses the evaluations points for Zippel's sparse interpolation such that the system of linear equations is a transposed Vandermonde system. Therefore fewer evaluation points are used and all linear systems can be solved efficiently in quadratic time, see Section 2.7.

It is possible that the prime p used in PGCD may divide a coefficient C_{ij} in $K_r(H)$ in which case we will need to call PGCD again with a different prime to compute more of the support of $K_r(H)$.

Definition 5.1. Let $f = \sum_{i=0}^{d} C_i y^{e_i}$ be a polynomial in $\mathbb{Z}[y]$. We say a prime p causes *missing terms* in f if p divides any coefficient C_i in f.

Our strategy to detect when $\text{Supp}(\widehat{h}_i) \not\subset \text{Supp}(K_r(h_i))$ is probabilistic. We use one extra equation and solve $(t_i + 1)$ by t_i systems thus requiring $t + 1$ evaluations instead of $2t + 2$. Once detected, we will call PGCD again to determine $\text{Supp}(K_r(h_i))$.

5.4 Algorithm MGCD1

We now present our algorithm as algorithm MGCD1 which calls subroutines PGCD1 and SGCD1. Like MGCD, MGCD1 loops calling PGCD1 to determine the $Hp = K_r(H) \mod p$. Instead of calling PGCD1 for each prime, MGCD1 after PGCD1 returns an image Hp, MGCD1 assumes the support of $K_r(H)$ is now known and uses SGCD1 for the remaining images.

Algorithm MGCD1(A, B)

Inputs $A, B \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$ satisfying $n > 0$, A and B are primitive in x_0, and $\deg_{x_0} A > 0$, $\deg_{x_0} B > 0$.

Output $G = \gcd(A, B)$.

1 If $\#LC(A) < \#LC(B)$ set $\Gamma = LC(B)$ else set $\Gamma = LC(A)$.

2 Call Algorithm DegreeBound(A,B,i) to get $d_i \geq \deg_{x_i} G$ for $0 \leq i \leq n$.
If $d_0 = 0$ **return 1**.

3 Set $r_i = \min(\deg_{x_i} A, \deg_{x_i} B, d_i + \deg_{x_i}(\Gamma))$ for $1 \leq i < n$.
Set $\delta = 1$.

Kronecker-Prime

4 Set $r_i = r_i + 1$ for $1 \leq i < n$. Let $Y = (y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{n-1}})$.
Set $K_r(A) = A(x, Y)$, $K_r(B) = B(x, Y)$ and $K_r(\Gamma) = \Gamma(Y)$.
If K_r is bad **goto** Kronecker-Prime otherwise set $\delta = \delta + 1$.

RESTART

5 Set $\widehat{H} = 0, M = 1$ and MissingTerms = true.
Set $\sigma_i = \phi$ and $\tau_i = 0$ for $0 \leq i \leq d_0$. // $\tau_i = |\sigma_i|$.

LOOP: // Invariant: $d_0 \geq \deg_{x_0} H$.

6 Compute $dx = DegreeBound(A, B, 0)$.
If $dx < d_0$ set $d_0 = dx$ and **goto** RESTART.

7 For each prime $p|M$ do // Delete bad image.

8 Set $a = K_r(A) \mod p$, $b = K_r(B) \mod p$ and $h = \widehat{H} \mod p$.

9 Pick β from $[0, p-1]$ uniformly at random.

10 If $K_r(\Gamma)(\beta) \neq 0$ and either $h(x, \beta) \not| a(x, \beta)$ or $h(x, \beta) \not| b(x, \beta)$ then h is wrong so set $M = M/p$ and $\widehat{H} = \widehat{H} \mod M$ to remove it.

End for loop.

If MissingTerms then

11 Pick a new smooth prime $p > 2^{\delta} \prod_{i=1}^{n} r_i$ that is not bad.

12 Call PGCD1($K_r(A)$, $K_r(B)$, $K_r(\Gamma)$, d_0, τ, r, p).

13 If PGCD1 returned UNLUCKY($dmin$) set $d_0 = dmin$ and **goto** RESTART.
If PGCD1 returned FAIL **goto** Kronecker-Prime.

14 Let $\widehat{H}p = \sum_{i=0}^{dx} \widehat{h}_i(y) x^i$ be the output of PGCD1.
Set MissingTerms = false, $\sigma_i := \sigma_i \cup \mathrm{Supp}(\widehat{h}_i)$ and $\tau_i = |\sigma_i|$ for $0 \leq i \leq d_0$.

else

15 Pick a new prime $p > 2^{\delta} \prod_{i=1}^{n} r_i$ that is not bad.

16 Call SGCD1($K_r(A)$, $K_r(B)$, $K_r(\Gamma)$, d_0, σ, τ, p).

17 If SGCD1 returned UNLUCKY($dmin$) set $d_0 = dmin$ and **goto** RESTART.
If SGCD1 returned FAIL **goto** Kronecker-Prime.
If SGCD1 returned MISSINGTERMS set $\delta = \delta + 1$, Missingterms = true and **goto** LOOP.

18 Let $\widehat{Hp} = \sum_{i=0}^{d_0} \widehat{h}_i(y)x^i$ be the output of SGCD1.

End If

Chinese-Remaindering

19 Set $Hold = \widehat{H}$. Solve $\{\widehat{H} \equiv Hold \mod M$ and $\widehat{H} \equiv \widehat{Hp} \mod p\}$ for $\widehat{H}$. Set $M = M \times p$. If $\widehat{H} \neq Hold$ then **goto** LOOP.

Termination.

20 Set $\widetilde{H} = K_r^{-1}\widehat{H}(x, y)$. Let $\widetilde{H} = \sum_{i=0}^{d_0} \widetilde{C}_i x_0^i$ where $\widetilde{C}_i \in \mathbb{Z}[x_1, \ldots, x_n]$.

21 Set $\widehat{G} = \widetilde{H}/\gcd(\widetilde{C}_0, \widetilde{C}_1, \ldots, \widetilde{C}_{d_0})$ ($\widehat{G}$ is the primitive part of $\widetilde{H}$.

22 If $\deg \widehat{G} \leq \deg A$ and $\deg \widehat{G} \leq \deg B$ and $\widehat{G}|A$ and $\widehat{G}|B$ then **return** $\widehat{G}$.

23 **goto** LOOP.

Algorithm PGCD1($K(A)$, $K(B)$, $K(\Gamma)$, d_0, τ, r, p)

Inputs $K(A), K(B) \in \mathbb{Z}[x, y]$ and $K(\Gamma) \in \mathbb{Z}[y]$, $d_0 \geq \deg_{x_0} G$ where $G = \gcd(A, B)$, term bound estimates $\tau \in \mathbb{Z}^{d_0+1}$, $r \in \mathbb{Z}^n$, and a smooth prime p.

Output $Hp \in \mathbb{Z}_p[x, y]$ satisfying $Hp = K(H) \bmod p$ or FAIL or UNLUCKY(dmin).

1 Pick a shift $s \in \mathbb{Z}_p^*$ uniformly at random and any generator α for $\mathbb{Z}_p^*$.

2 Set $T = 0$.

LOOP

3 For j from $2T$ to $2T + 1$ do

4 Compute $a_j = K(A)(x, \alpha^{s+j}) \bmod p$ and $b_j = K(B)(x, \alpha^{s+j}) \bmod p$.

5 If $\deg_x a_j < \deg_x K(A)$ or $\deg_x b_j < \deg_{x_0} K(B)$ then return FAIL (α^{s+j} is a bad evaluation point.)

6 Compute $g_j = \gcd(a_i, b_i) \in \mathbb{Z}_p[x]$ using the Euclidean algorithm.
Make g_j monic and set $g_j = K(\Gamma)(\alpha^{s+j}) \times g_j \bmod p$.

End for loop.

7 Set $dmin = \min \deg g_j(x)$ and $dmax = \max \deg g_j$ for $2T \le j \le 2T+1$.
If $dmin < d_0$ **output** UNLUCKY(dmin).
If $dmax > d_0$ **output** FAIL (unlucky evaluations).

8 Set $T = T + 1$.
If $T < \#K(\Gamma)$ or $T < \max_{i=0}^{d_0} \tau_i$ **goto** LOOP.

9 For i from 0 to d_0 do

10 Run the Berlekamp-Massey algorithm on the coefficients of x^i in the images $g_0, g_1, \ldots, g_{2T-1}$ to obtain $c_i(z)$ and set $\tau_i = \deg c_i(z)$. If either of the last two discrepancies were non-zero **goto** LOOP.

End for loop.

11 For i from 0 to d_0 do

12 Compute the reciprocal of $c_i(z)$ to get $\Lambda_i(z)$ then compute the roots of $\Lambda_i(z)$. If $\Lambda_i(0) = 0$ or the number of distinct roots of $\Lambda_i(z)$ is not equal τ_i then **goto** LOOP ($c_i(z)$ stabilized too early)

13 Set $e_k = \log_\alpha m_k$ for $1 \le k \le \tau_i$ and let $\sigma_i = \{y^{e_1}, y^{e_2}, \ldots y^{e_{\tau_i}}\}$.
If $e_k \ge \prod_{i=1}^{n} r_i$ then $e_k > \deg K_r(H)$ so **output** FAIL (either the $c_i(z)$ stabilized too early or K_r or p or all evaluations are unlucky).

14 Solve the τ_i by τ_i shifted transposed Vandermonde system

$$\left\{ \sum_{k=1}^{\tau_i} (\alpha^{s+j})^{e_k} u_k = \text{coefficient of } x^i \text{ in } g_j(x) \ \ \textbf{for } 0 \le j < \tau_i \right\}$$

modulo p for u and set $\widehat{h}_i(y) = \sum_{k=1}^{\tau_i} u_k y^{e_k}$. Note: $(\alpha^{s+j})^{e_k} = m_k^{s+j}$.

End for loop.

15 Set $Hp = \sum_{i=0}^{d_0} \widehat{h}_i(y) x^i$ and **output** Hp.

Algorithm SGCD1($K(A)$, $K(B)$, $K(\Gamma)$, d_0, σ, τ, p)

Inputs $K(A), K(B) \in \mathbb{Z}[x,y]$, $K(\Gamma) \in \mathbb{Z}[y]$, $d_0 \ge \deg_{x_0} G$ where $G = \gcd(A,B)$, supports σ_i for $K_r(h_i)$ and $\tau_i = |\sigma_i|$, a smooth prime p.

Output FAIL or UNLUCKY(dmin) or MISSINGTERMS or $Hp \in \mathbb{Z}_p[x,y]$ satisfying if $d_0 = \deg_{x_0} G$ and $\sigma_i = \text{Supp}(K_r(h_i))$ then $Hp = K_r(H) \mod p$.

1 Pick a shift $s \in \mathbb{Z}_p^*$ uniformly at random and any generator α for $\mathbb{Z}_p^*$.

2 Set $T = \max_{i=1}^{d_0} \tau_i$.

3 For j from 0 to T do // includes 1 check point

4 Compute $a_j = K(A)(x, \alpha^{s+j}) \bmod p$ and $b_j = K(B)(x, \alpha^{s+j}) \bmod p$.

5 If $\deg_x a_j < \deg_x K(A)$ or $\deg_x b_j < \deg_{x_0} K(B)$ then **output** FAIL (α^{s+j} is a bad evaluation point.)

6 Compute $g_j = \gcd(a_i, b_i) \in \mathbb{Z}_p[x]$ using the Euclidean algorithm.
Make g_j monic and set $g_j = K(\Gamma)(\alpha^{s+j}) \times g_j \bmod p$.

End for loop.

6 Set $dmin = \min \deg g_j(x)$ and $dmax = \max \deg g_j$ for $0 \le j \le T$.
If $dmin < d_0$ **output** UNLUCKY($dmin$).
If $dmax > d_0$ **output** FAIL (unlucky evaluations).

7 For i from 0 to d_0 do

8 Let $\sigma_i = \{y^{e_1}, y^{e_2}, \ldots, y^{e_{\tau_i}}\}$. Solve the $(\tau_i + 1)$ by τ_i shifted transposed Vandermonde system

$$\left\{ \sum_{k=1}^{\tau_i} (\alpha^{s+j})^{e_k} u_k = \text{coefficient of } x^i \text{ in } g_j(x) \quad \textbf{for } 0 \le j \le \tau_i \right\}$$

modulo p for u.

9 If the linear system is inconsistent then **output** MISSINGTERMS, otherwise set $\widehat{h}_i(y) = \sum_{k=1}^{\tau_i} u_k y^{e_k}$.

End for loop.

10 Set $Hp = \sum_{i=0}^{d_0} \widehat{h}_i(y) x^i$ and **output** Hp.

Theorem 5.1. *MGCD1 terminates and outputs* $G = \gcd(A, B)$.

Proof. Since MGCD1 avoids bad Kronecker substitutions and bad primes, PGCD1 and SGCD1 also avoid bad evaluation points α^{s+j}, we have $K_r(\Gamma)(\alpha^{s+j}) \neq 0$ and $\deg g_j(x) \ge \deg_{x_0} G$ by Lemma 2.6. Hence $\deg_x \widehat{H} = \deg_{x_0} \widehat{G} \ge \deg_{x_0} G$. Therefore, if algorithm MGCD1 terminates, the conditions A and B are primitive and $\widehat{G}|A$ and $\widehat{G}|B$ imply $\widehat{G} = G$. To prove the termination we observe that Algorithm MGCD1 proceeds in four phases.

In the first phase MGCD1 loops while $d_0 > \deg_x K_r(H) = \deg_{x_0} G$. Because Γ is either $LC(A)$ or $LC(B)$, even if K_r or p or all evaluation points are unlucky, the scaled images in Step 6 of algorithm PGCD1 are images of a polynomial in $\mathbb{Z}[x, y]$ hence the $c_i(z)$ polynomials stabilize with high probability, see Theorem 2.6, and algorithm PGCD1 always terminates.

Now if PGCD1 or SGCD1 output UNLUCKY($dmin$) then d_0 is decreased, otherwise, they output FAIL or MISSINGTERMS or an image Hp and MGCD1 executes Step 7 at the beginning of the main loop. Eventually the call to DegreeBound in Step 7 will set $d_0 = \deg_{x_0} G$ after which unlucky Kronecker substitutions, unlucky primes and unlucky evaluation points can be detected. Hence d_0 is the key parameter in MGCD1. In practice, DegreeBound determines the correct d_0 in one run if the prime used is large enough.

Suppose $d_0 = \deg_{x_0} G$ for the first time. In the second phase MGCD1 loops while PGCD1 outputs FAIL due to an unlucky Kronecker substitution or an unlucky prime or bad or unlucky evaluation points or the Berlekamp-Massey algorithm stabilized too early. If PGCD1 outputs FAIL, since we don't know if this is due to unlucky Kronecker substitution or an unlucky prime p, MGCD1 increases r_i by 1 and the size of p by 1 bit. Since there are only finitely many unlucky K_r, eventually K_r will be lucky. And since there are only finitely many unlucky primes, eventually p be lucky. Finally, since we keep increasing the length of p, eventually p will be sufficiently large so that no bad or unlucky evaluations are encountered in PGCD1 and the Berlekamp-Massey algorithm does not stabilize too early. Then PGCD1 succeeds and outputs an image Hp with $\deg_x Hp = d_0 = \deg_{x_0} G$.

In the third phase MGCD1 loops while $\sigma_i \not\supseteq K_r(h_i)$, that is, we don't yet have the support for all $K_r(h_i) \in \mathbb{Z}[y]$ either because of missing terms or because a $c_i(z)$ polynomial stabilized too early in PGCD1, and went undetected.

We now prove that SGCD1 detects that $\sigma_i \not\supseteq \text{Supp}(K_r(h_i))$ with probability at least 3/4 in Step 22 so that PGCD1 is called again in MGCD1.

Suppose $\sigma_i \not\supseteq \text{Supp}(K_r(h_i))$ for some i. Consider the first τ_i equations in Step 8 of SGCD1. We first argue that this linear system has a unique solution. Let $m_k = \alpha^{e_k}$ so that $(\alpha^{s+j})^{e_k} = m_k^{s+j}$. The coefficient matrix W of the linear system has entries

$$W_{jk} = m_k^{s+j-1} \text{ for } 1 \le j \le \tau_i \text{ and } 1 \le k \le \tau_i.$$

W is a shifted transposed Vandermonde matrix with determinant

$$\det W = \prod_{1 \le k \le \tau_i} m_k^s \prod_{1 \le j < k \le \tau_i} (m_j - m_k).$$

Since $m_k = \alpha^{e_k}$ we have $m_k \neq 0$. Since $p > \deg_y K_r(H)$ and the number of distinct roots of $\Lambda_i(z)$ equals τ_i (no multiple roots) in Step 9 of PGCD1, the m_k are distinct. Hence $\det W \neq 0$ and the linear system has a unique solution for u.

Let $\bar{h}_i(y) = \sum_{k=1}^{\tau_i} u_k y^{e_k}$ and let $E(y) = K_r(h_i)(y) - \bar{h}_i(y)$. If the $\tau_i + 1$ by τ_i linear system in Step 8 of SGCD1 is inconsistent then $E(\alpha^{s+j}) = 0$ for $0 \le j < \tau_i$ but $E(\alpha^{s+\tau_i}) \neq 0$ and algorithm SGCD1 detects the inconsistency. It is possible, however, that $E(\alpha^{s+\tau_i}) = 0$ and algorithm SGCD1 fails to detect the inconsistency. But the probability that this can happen is at most 1/4, see Lemma 5.1 below.

Thus eventually the wrong support is detected in Step 8 of algorithm SGCD1. Because we cannot tell whether this is caused by missing terms or $c_i(z)$ stabilizing too early and going undetected in Steps 12 and 13 of PGCD1, we increase the size of p by 1 bit in Step 17 so that with repeated calls to PGCD1, $c_i(z)$ will eventually not stabilize early and we obtain $\sigma_i \supseteq \text{Supp}(K_r(h_i))$.

How many good images are needed before $\sigma_i \supseteq \text{Supp}(K_r(h_i))$ for all $0 \le i \le d_0$? Let *pmin* be the smallest prime used by algorithm PGCD1. Let $N = \lfloor \log_{pmin} \|K_r(H)\| \rfloor$. Since at most N primes $\ge pmin$ can divide any integer coefficient in $K_r(H)$ then $N+1$ good images from PGCD1 are sufficient to recover the support of $K_r(H)$.

In the fourth and final phase MGCD1 loops calling SGCD1 while $\widehat{H} \neq K_r(H)$. If SGCD1 outputs an image Hp then $Hp = H \mod p$ because $d_0 = \deg_{x_0} H$ and $\sigma_i \supseteq K_r(h_i)$. The image is combined with previously computed images in $\widehat{H}$ using Chinese remaindering. But as noted in Example 5.1, $\widehat{H}$ may contain a bad image. A bad image arises because either PGCD1 returns a bad image Hp due to early stabilization of $c_i(z)$ or because SGCD1 uses a support with missing terms and fails to detect it.

Consider the prime p and polynomial $h(x,y)$ in step 8 of MGCD1. Suppose $h(x,y)$ is a bad image, that is, $h \neq K_r(H) \mod p$. We claim steps $7-10$ of MGCD1 detect this bad image with probability at least $1/2$. Since the test for a bad image is executed repeatedly in the main loop, algorithm MGCD1 eventually detects it and removes it hence eventually MGCD1 computes $K_r(H)$ and terminates with output G.

To prove the claim we recall that $H = \Delta G$ and $LC(H) = \Gamma$. Because step 8 of PGCD1 requires $T \ge \#K_r(\Gamma)$ this ensures algorithm PGCD1 always outputs Hp with $LC(Hp) = K_r(\Gamma) \mod p$, hence $LC(h) = K_r(\Gamma) \mod p$.

If $h = K_r(H) \mod p$ and $K_r(\Gamma)(\beta) \neq 0$ then in step 10 of MGCD1 $h(x,\beta)$ must divide $K_r(A)(x,\beta)$ and divide $K_r(B)(x,\beta)$. Now suppose $h \neq K_r(H) \mod p$. Then step 10 of MGCD1 fails to detect this bad image if $K_r(\Gamma)(\beta) \neq 0$ and $h(x,\beta) | K_r(A)(x,\beta)$ and $h(x,\beta) | K_r(B)(x,\beta)$ in $\mathbb{Z}_p[x]$. Since $\deg_x h = d_0 = \deg_x K_r(H)$ it must be that $h(x,\beta)$ is an associate of $K_r(H)(x,\beta)$. But since $LC(h) = K_r(\Gamma) \mod p = LC(K_r(H)) \mod p$ we have $h(x,\beta) = K_r(H)(x,\beta) \mod p$. Let $E = h - K_r(H) \mod p$. Therefore the test for a bad image h succeeds if $K_r(\Gamma)(\beta) \neq 0$ and $E(x,\beta) \neq 0$. Lemma 5.2 below implies the test succeeds with probability at least $1/2$.

□

Lemma 5.1. *We use the notations defined in the proof of Theorem 5.1. If s is chosen uniformly at random from $[1, p-1]$ then*

$$\text{Prob}[E(\alpha^{s+\tau_i}) = 0] \ < \ \frac{1}{4}.$$

Proof. The condition in Step 13 of algorithm PGCD1 means $\deg \bar{h}_i(y) < \prod_{j=1}^n r_j$ hence $\deg_y(E) < \prod_{j=1}^n r_j$. Now s is chosen uniformly at random so $\alpha^{s+\tau_i}$ is random on $[1, p-1]$ therefore

$$\text{Prob}[E(\alpha^{s+\tau_i}) = 0] \le \frac{\deg_y(E)}{p-1} < \frac{\prod_{j=1}^n r_j}{p-1}.$$

Since the primes in SGCD1 satisfy $p > 4 \prod_{j=1}^n r_j$ the result follows. □

Lemma 5.2. *We use the notations defined in the proof of Theorem 5.1. If β is chosen uniformly at random from $[0, p-1]$ then*

$$Prob[K_r(\Gamma)(\beta) \neq 0] \geq \frac{3}{4} \text{ and } Prob[E(x,\beta) \neq 0] \geq \frac{3}{4}.$$

Proof. The primes p chosen in step 15 of MGCD1 satisfy $p > 2^\delta \prod_{i=1}^n r_i$ with $\delta \geq 2$. Since $\deg_y K_r(\Gamma) < \prod_{i=1}^n r_i$ by step 3 of MGCD1, $Prob[K_r(\Gamma)(\beta) \mod p = 0] \leq \frac{\deg_y(\Gamma)}{p} < \frac{1}{4}$. Since $\deg_y h < \prod_{i=1}^n r_i$ by step 13 of PGCD1 and since r_i is chosen in step 3 of MGCD1 so that $r_i \geq \deg_{x_i} H$, we have $\deg_y K_r(H) < \prod_{i=1}^n r_i$. Hence $\deg_y E < \prod_{i=1}^n r_i$. Therefore $\text{Prob}[E(x,\beta) = 0] \leq \frac{\deg_y E}{p} < \frac{1}{4}$. □

Please note that we only require to have $\sigma_i \supseteq \text{Supp}(K_r(h_i))$ eventually instead of $\sigma_i = \text{Supp}(K_r(h_i))$. Hence there may be wrong monomials in σ_i. However, this does not affect the correctness of our algorithm. Now suppose $\sigma_i \supseteq \text{Supp}(K_r(h_i))$. Step 7 - 10 in MGCD1 detects bad images and remove them, but we don't have to remove the wrong monomials the bad images introduced in step 14 of MGCD1 because the coefficients of those wrong monomials must be zero in the output of SGCD1 if the linear system is consistent. Let's consider the following example:

Example 5.3. *Consider the GCD problem $G = H = x^3 + (C_1y^3 + C_2y)x + (C_3y^4 + C_4y^2) \in \mathbb{Z}[x,y]$ where C_1, C_2, C_3, C_4 are constants and $\bar{A} = 1, \bar{B} = 1$. Suppose the first call of PGCD1 with prime p_1 returns $\widehat{H}_{p_1} = x^3 + (\cdot y^2)x + (\cdot y^4 + \cdot y^2)$ where $\cdot$ denotes non-zero constant hence $\sigma_1 = \{y^2\}$ and $\sigma_0 = \{y^2, y^4\}$. Obviously $\sigma_1 = \{y^2\}$ is wrong due to the feedback polynomial $c_1(z)$ stabilizing too early. $\widehat{H}_{p_1}$ should be removed by the checking step 7-10 in MGCD1. But we don't reset σ_1 and σ_0 to be empty. Then suppose the call of SGCD1 with prime p_2 detects the inconsistency and returns MISSINGTERMS. We have to call PGCD1 again with prime p_3 and suppose the feedback polynomial stabilizes too early again and we get $\widehat{H}_{p_3} = x^3 + (\cdot y^3 + \cdot y)x + \cdot y^3$. It is clear that the support of the coefficient of x^0 is wrong. The Step 14 in MGCD1 combines the results from those 2 calls of PGCD1 and get $\sigma_1 = \{y, y^2, y^3\}$ and $\sigma_0 = \{y^2, y^3, y^4\}$. $\widehat{H}_{p_3}$ should be removed by the checking step 7-10 in MGCD1. The next call of SGCD1 with prime p_4 should return $\widehat{H}_{p_4} = x^3 + (\cdot y^3 + 0y^2 + \cdot y)x + (\cdot y^4 + 0y^3 + \cdot y^2) = x^3 + (\cdot y^3 + \cdot y)x + (\cdot y^4 + \cdot y^2)$ which is a good image modulo p_4. Then we just use σ_1 and σ_0 to compute good images as many as we need to successfully lift the coefficients of $\widehat{H}$ to a desired size so that $\widehat{H} = K_r(H)$. In this example, all images computed by PGCD1 are partially correct (some monomials are correct and some are not) and σ_1, σ_0 are confirmed by SGCD1 with p_4. In H_{p_4}, we can only say that y^2 and y^3 with zero coefficients may not be in σ_1 and σ_0 respectively because p_1 and p_3 may just be unlucky primes and divide their coefficients. In this case if we remove y^2, y^3 from σ_1 and σ_0, we might never get the correct support. That's the reason we only remove bad images but keep possibly wrong monomials in the support.*

Chapter 6

Implementation

We observed that most of time in our GCD algorithm was in evaluation so here we consider how to improve the evaluation algorithm and also how to reduce t the number of evaluation points needed to interpolate H. We also present some timing results comparing our new algorithm which we have implemented in Cilk C with the Maple and Magma C implementations of Zippel's GCD algorithm.

6.1 Evaluation

Let $A, B \in \mathbb{Z}_p[x_0, \ldots, x_n]$, $d = \max_{i=1}^{n} d_i$ where $d_i = \max(\deg_{x_i} A, \deg_{x_i} B)$. If we use a Kronecker substitution

$$K(A) = A(x, y, y^{r_1}, \ldots, y^{r_1 r_2 \cdots r_{n-1}}) \text{ with } r_i = d_i + 1,$$

then $\deg_y K(A) < (d+1)^n$. Let $s = \#A + \#B$, we can evaluate the s monomials in $K(A)(x, y)$ and $K(B)(x, y)$ at $y = \alpha^k$ in $O(sn \log d)$ multiplications. Instead we first compute $\beta_1 = \alpha^k$ and $\beta_{i+1} = \beta_i^{r_i}$ for $i = 1, 2, \ldots, n-1$ then precompute n tables of powers $1, \beta_i, \beta_i^2, \ldots, \beta_i^{d_i}$ for $1 \le i \le n$ using at most nd multiplications. Now, for each term in A and B of the form $c x_0^{e_0} x_1^{e_1} \ldots x_n^{e_n}$ we compute $c \times \beta_1^{e_1} \times \cdots \times \beta_n^{e_n}$ using the tables in n multiplications. Hence we can evaluate $K(A)(x, \alpha^k)$ and $K(B)(x, \alpha^k)$ in at most $nd + ns$ multiplications. Thus for T evaluation points $\alpha, \alpha^2, \ldots, \alpha^T$, the evaluation cost is $O(ndT + nsT)$ multiplications.

When we first implemented the algorithm PGCD, we noticed that often well over 95% of the time was spent to evaluate the input polynomials A and B at the points α^k. This happens when $\#G \ll \#A + \#B$. The following method in Table 6.1 uses the fact that for a monomial $M_i(x_1, x_2, \ldots, x_n)$

$$M_i(\beta_1^k, \beta_2^k, \ldots, \beta_n^k) \;=\; M_i(\beta_1, \beta_2, \ldots, \beta_n)^k$$

to reduce the total evaluation cost from $O(ndT + nsT)$ multiplications to $O(nd + ns + sT)$. Note that no sorting on x_0 is needed in step 4b if the monomials in the input A are sorted on x_0.

Algorithm Evaluate.
Input $A = \sum_{i=1}^{m} c_i x_0^{e_i} M_i(x_1, \ldots, x_n) \in \mathbb{Z}_p[x_0, \ldots, x_n]$, $T > 0$, $\beta_1, \beta_2, \ldots, \beta_n \in \mathbb{Z}_p$, and integers $d_1, d_2, \ldots, d_n$ with $d_i \geq \deg_{x_i} A$.
Output $y_k = A(x_0, \beta_1^k, \ldots, \beta_n^k)$ for $1 \leq k \leq T$.

1 Create the vector $C = [c_1, c_2, \ldots, c_m] \in \mathbb{Z}_p^m$.
2 Compute $[\beta_i^j : j = 0, 1, \ldots, d_i]$ for $1 \leq i \leq n$.
3 Compute $\Gamma = [M_i(\beta_1, \beta_2, \ldots, \beta_n) : 1 \leq i \leq m]$.
4 For $k = 1, 2, \ldots, T$ do
  4a Compute the vector $C := [C_i \times \Gamma_i \text{ for } 1 \leq i \leq m]$.
  4b Assemble $y_k = \sum_{i=1}^{m} C_i x_0^{e_i} = A(x_0, \beta_1^k, \ldots, \beta_n^k)$.

Figure 6.1: Matrix evaluation algorithm

The algorithm in Figure 6.1 computes y_k as the matrix vector product.

$$\begin{bmatrix} \Gamma_1 & \Gamma_2 & \ldots & \Gamma_m \\ {\Gamma_1}^2 & {\Gamma_2}^2 & \ldots & {\Gamma_m}^2 \\ \vdots & \vdots & \vdots & \vdots \\ {\Gamma_1}^T & {\Gamma_2}^T & \ldots & {\Gamma_m}^T \end{bmatrix} \begin{bmatrix} c_1 x_0^{e_1} \\ c_2 x_0^{e_2} \\ c_3 x_0^{e_3} \\ \vdots \\ c_m x_0^{e_m} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_T \end{bmatrix}.$$

Even with this improvement evaluation still takes most of the time so we must parallelize it. Each evaluation of A could be parallelized in blocks of size m/N for N cores. In Cilk C, this is only effective, however, if the blocks are large enough (at least 50,000) so that the time for each block is much larger than the time it takes Cilk to create a task. For this reason, it is necessary to also parallelize on k. To parallelize on k for N cores, we multiply the previous N values of C in parallel by the vector

$$\Gamma_N = [M_i(\beta_1, \beta_2, \ldots, \beta_n)^N : 1 \leq i \leq m].$$

Because most of the time is still in evaluation, Monagan and Wong [Monagan and Wong, 2017] developed a fast parallel multi-point evaluation algorithm, which is based on the fast multi-point evaluation technique described by van der Hoven and Lecerf [van der Hoven and Lecerf, 2013], for our fast parallel GCD algorithm. This new algorithm has complexity $O(nd + ns + s \log^2 T)$ which is better than $O(nd + ns + sT)$ if T is large. Let $C = [c_1 x_0^{e_1}, c_2 x_0^{e_2}, \ldots, c_m x_0^{e_m}]$. We consider to evaluate A. The fast multi-point evaluation

algorithm is based on the observation

$$(1-\Gamma_i u)^{-1} = 1 + \Gamma_i u + \Gamma_i^2 u^2 + \Gamma_i^3 u^3 \cdots .$$

We define the function $f(u)$ and expand it.

$$\begin{aligned} f(u) &= \frac{C_1}{1-\Gamma_1 u} + \frac{C_2}{1-\Gamma_2 u} + \cdots + \frac{C_m}{1-\Gamma_m u} \\ &= C_1(1+\Gamma_1 u + \Gamma_1^2 u^2 + \cdots) + \cdots + C_m(1+\Gamma_m u + \Gamma_m^2 u^2 + \cdots) \\ &= (C_1 + \cdots + C_m) + (C_1\Gamma_1 + \cdots + C_m\Gamma_m)u + (C_1\Gamma_1^2 + \cdots + C_m\Gamma_m^2)u^2 + \cdots \\ &= y_0 + y_1 u + y_2 u^2 + \cdots + y_T u^T + \cdots . \end{aligned}$$

Now we can split the sum f into $s = \lceil m/T \rceil$ blocks, then we have $f = B_1(u) + B_2(u) + \cdots + B_s(u)$. For each $B_k(u)$, we use the divide-and-conquer strategy to put terms in each block over a common denominator, then expand $B_k(u)$ up to $O(u^T)$ to get the coefficients by using the fast series inversion. The total number of multiplications to compute f is $O(\lceil m/T \rceil M(T) \log T)$ where $M(T)$ denotes the cost of dense univariate polynomial multiplications of degree T. With Fast Fourier transform, $M(T) = O(T \log T)$. Since $m < s$, the total cost to obtain f is $O(s \log^2 T)$ which replaces $O(sT)$ in the matrix evaluation algorithm. The divide-and-conquer method computes f recursively from top to bottom with T provided. But in our application we don't know T in advance. Hence we first pick a relatively small T and compute f from bottom to top. See [Monagan and Wong, 2017, Figure 1], they pick $T = 2^k$ for $k = 0, 1, 2, \ldots$.

6.2 The non-monic case and homogenization

Algorithm PGCD interpolates $H = \Delta G$ from scaled monic images $K(\Gamma)(\alpha^j)g_j(x)$ which are computed in Steps 9 and 12a. If the number of terms of Δ is m and $m > 1$ then it is likely that $\#H$ is greater than $\#G$, which means we need more evaluation points for sparse interpolation. For sparse inputs, this may increase t by a factor of m.

One such example occurs in multivariate polynomial factorization. Given a polynomial f in $\mathbb{Z}[x_0, x_1, \ldots, x_n]$, factorization algorithms first identify and remove repeated factors by doing a square-free factorization. See Section 8.1 of [Geddes et al., 1992]. The first step of square-free factorization computes

$$g = \gcd(f, h = \frac{\partial f}{\partial x_0}).$$

Then we have

$$\Gamma = \gcd(LC(f), LC(h)) = \gcd(LC(f), dLC(f)) = LC(f) \quad \text{and} \quad \Delta = LC(f)/LC(g)$$

which can be a large polynomial.

Obviously, if either A or B is monic in x_i for some $i > 0$ then we may simply use x_i as the main variable in our GCD algorithm instead of x_0 so that $\#\Gamma = \#\Delta = 1$. Similarly, if either A or B have a constant term in any x_i, that is, $A = \sum_{j=0} a_j x_i^j$ and $B = \sum_{j=0} b_j x_i^j$ and either a_0 or b_0 are integers, then we can reverse the coefficients of both A and B in x_i so that again $\#\Gamma = \#\Delta = 1$. But many multivariate GCD problems in practice do not satisfy any of these conditions.

Suppose A or B has a constant term. We propose to exploit this by homogenizing A and B. Let f be a non-zero polynomial in $\mathbb{Z}[x_1, x_2, \ldots, x_n]$ and

$$H_z(f) = f(\frac{x_1}{z}, \frac{x_2}{z}, \ldots, \frac{x_n}{z}) z^{\deg f}$$

denote the homogenization of f in z. We have the following properties of $H_z(f)$.

Lemma 6.1. *Let a and b be in $\mathbb{Z}[x_1, x_2, ..., x_n]$. For non-zero a and b*

(i) $H_z(a)$ is homogeneous in $z, x_1, \ldots, x_n$ of degree $\deg a$,
(ii) $H_z(a)$ is invertible: if $f(z) = H_z(a)$ then $H_z^{-1}(f) = f(1) = a$,
(iii) $H_z(ab) = H_z(a)H_z(b)$, and
(iv) $H_z(\gcd(a, b)) = \gcd(H_z(a), H_z(b))$.

Proof: To prove (i) let $M = x_1^{d_1} x_2^{d_2} \ldots x_n^{d_n}$ be a monomial in a and let $d = \deg a$. Then

$$H_z(M) = z^d \frac{x_1^{d_1}}{z^{d_1}} \cdots \frac{x_n^{d_n}}{z^{d_n}}.$$

Observe that since $d \geq d_1 + d_2 + \cdots + d_n$ then $\deg_z(H_z(M)) \geq 0$ and $\deg H_z(M) = d$. Properties (ii) and (iii) follow easily from the definition of H_z. To prove (iv) let $g = \gcd(a, b)$. Then $a = g\bar{a}$ and $b = g\bar{b}$ for some $\bar{a}, \bar{b}$ with $\gcd(\bar{a}, \bar{b}) = 1$. Now

$$\begin{aligned} \gcd(H_z(a), H_z(b)) &= \gcd(H_z(g\bar{a}),\ H_z(g\bar{b})) \\ &= \gcd(H_z(g)H_z(\bar{a}),\ H_z(g)H_z(\bar{b})) \text{ by (iii)} \\ &= H_z(g) \times \gcd(H_z(\bar{a}),\ H_z(\bar{b})) \text{ up to units.} \end{aligned}$$

Let $c(z) = \gcd(H_z(\bar{a}), H_z(\bar{b}))$ in $\mathbb{Z}[z, x_1, \ldots, x_n]$. It suffices to prove that $\gcd(\bar{a}, \bar{b}) = 1$ implies $c(z)$ is a unit. Now $c(z) = \gcd(H_z(\bar{a}), H_z(\bar{b})) \Rightarrow c(z) | H_z(\bar{a})$ and $c(z) | H_z(\bar{b})$ which implies

$$H_z(\bar{a}) = c(z)q(z) \text{ and } H_z(\bar{b}) = c(z)r(z)$$

for some $q, r \in \mathbb{Z}[z, x_1, \ldots, x_n]$. Applying H^{-1} to these relations we get $\bar{a} = c(1)q(1)$ and $\bar{b} = c(1)r(1)$. Now $\gcd(\bar{a}, \bar{b}) = 1$ implies $c(1)$ is a unit and thus $q(1) = \pm\bar{a}$ and $r(1) = \pm\bar{b}$. We need to show that $c(z)$ is a unit. Let $d = \deg H_z(\bar{a})$. Since $\deg H_z(\bar{a}) = \deg \bar{a}$ by (i) and $q(1) = \pm\bar{a}$ then $\deg q(1) = d$ and hence $\deg q(z) \geq d$. Now since $H_z(\bar{a}) = c(z)q(z)$ it

must be that $\deg c(z) = 0$ and $\deg q(z) = d$. Since $c(1) = \pm 1$ then $\deg c(z) = 0$ implies $c(z) = \pm 1$. $\square$.

Properties (iii) and (iv) mean we can compute $G = \gcd(A, B)$ using

$$G = H_z^{-1} \gcd(H_z(A), H_z(B)).$$

Notice also that homogenization preserves sparsity. To see why homogenization may help we consider an example.

Example 6.1. *Let* $G = x^2+y+1$, $\bar{A} = xy+x+y+1 = (y+1)x+(y+1) = (x+1)y+(x+1)$ *and* $\bar{B} = x^2y + xy^2 + x^2 + y^2 = (y+1)x^2 + y^2(x+1)$. *Then* $H_z(G) = z^2 + yz + x^2$, $H_z(\bar{A}) = z^2 + (x+y)z + xy$, *and* $H_z(\bar{B}) = (x^2+y^2)z + (x^2y + xy^2)$.

Notice in Example 6.1 that A and B are neither monic in x nor monic in y but since A has a constant term 1, $H_z(A)$ is monic in z. If we use x as x_0 in Algorithm PGCD then $\Gamma = \gcd(y+1, y+1) = y+1 = \Delta$ and we interpolate $H = \Delta G = (y+1)x^2 + (y^2+2y+1)$ and $t = 3$. If we use y as x_0 in Algorithm PGCD then $\Gamma = \gcd(x+1, x+1) = x+1 = \Delta$ and we interpolate $H = \Delta G = (x+1)y + (x^3+x^2+x+1)$ and $t = 4$. But if we use use z as x_0 in Algorithm PGCD then $\Gamma = \gcd(1, x^2+y^2) = 1$ hence $\Delta = 1$ and we interpolate $H_z(G) = z^2 + yz + x^2$ and $t = 1$.

If A or B has a constant term then because homogenizing A and B means $\Gamma = \Delta = c$ where c is a constant we always do this unless $\#\Gamma > 1$. There is, however, a cost to homogenizing for the GCD problem, namely, we increase the number of variables to interpolate by 1 and we increase the cost of the univariate images in $\mathbb{Z}_p[z]$ if the degree increases. The degree may increase by up to a factor of n as we see in the following example.

Example 6.2. *Let* $G = 1+\prod_{i=0}^{n} x_i^{d-1}$, $\bar{A} = 1+\prod_{i=0}^{n} x_i$ *and* $\bar{B} = 1-\prod_{i=0}^{n} x_i$ *then* $\deg_{x_i} A = d = \deg_{x_i} B$ *but* $\deg_z H_z(A) = nd = \deg_z H_z(B)$.

Homogenizing can also increase t when G has many terms of the same total degree. The following example demonstrates this case.

Example 6.3. *Let* $G = x^5 + (v+y)x^2 + (u^2+uy)x + vu^2 + 1$, $\bar{A} = (y+1)x + 1$ *and* $\bar{B} = (y+1)x+2$. *Let* x *be the main variable and if we use* $LC(A) = LC(B) = y+1$ *to scale* G, *then* $H = (y+1)x^5 + (vy+y^2+v+y)x^2 + (u^2y+uy^2+u^2+uy)x + (u^2vy+vu^2+y+1)$. *Hence* $t = 4$. *The homogenizing of* G *is* $H_z(G) = z^5 + (u^2v+u^2x+uxy+vx^2+x^2y)z^2 + x^5$, $H_z(\bar{A}) = z^2 + xz + xy$ *and* $H_z(\bar{B}) = 2z^2 + xz + xy$. *We use* $LC(H_z(A)) = 1$ *to scale* $H_z(G)$ *and find* $t = 5$. *Hence homogenizing in this example increases the number of variables and* t.

For our benchmark problem where $\Delta = 1$, we did, however, observe a 10% speedup. The reason is that the value for t decreased from 1198 to 1094.

6.3 Bivariate images

In my thesis proposal, I implemented our algorithm in Maple by mapping the inputs to bivariate polynomials then compute bivariate GCD images. For simplicity, our simplified and faster algorithms compute univariate GCD images in PGCD. With some minor modifications, our algorithm works with bivariate GCD images.

Recall that we interpolate $H = \sum_{i=0}^{dG} h_i(x_1, \ldots, x_n)x_0^i$ where $H = \Delta G$. The number of evaluation points used by algorithm PGCD is $2t + O(1)$ where $t = \max_{i=0}^{dG} \#h_i$. Since our implementation suggests that the evaluation dominates the cost of PGCD which is multiplied by the number of evaluation points needed, the cost of algorithm should be reduced if we can use smaller t.

Algorithm PGCD interpolates H from univariate images in $\mathbb{Z}_p[x_0]$. If instead we interpolate H from bivariate images in $\mathbb{Z}_p[x_0, x_1]$, this will likely reduce t when $\#\Delta = 1$ and when $\#\Delta > 1$. For our benchmark problem, where $\Delta = 1$, doing this reduces t from 1198 to 130 saving a factor of 9.2. On the other hand, we must now compute bivariate GCDs in $\mathbb{Z}_p[x_0, x_1]$. To decide whether this will lead to an overall gain, we need to know the cost of computing bivariate images and the likely reduction in t.

To compute a bivariate GCD in $\mathbb{Z}_p[x_0, x_1]$ we have implemented Brown's dense modular GCD algorithm from [Brown, 1971]. If G is sparse, then for sufficiently large t and n, G is likely dense in x_0 and x_1, so using a dense GCD algorithm is efficient. The complexity of Brown's algorithm is $O(d^3)$ arithmetic operations in $\mathbb{Z}_p$ where $d = \max_{i=0}^{1}(\deg_{x_i} A, \deg_{x_i} B)$. Thus if this cost is less than the cost of evaluating the inputs, which using our evaluation algorithm from 3.2 is s multiplications in $\mathbb{Z}_p$ where $s = \#A + \#B$, then the cost of the bivariate images does not increase the overall cost of the algorithm significantly. For our benchmark problem, $s = 2 \times 10^6$ and $d^3 = 40^3 = 64,000$ so the cost of a bivariate image is negligible compared with the cost of an evaluation.

Obviously, if interpolating H from bivariate images reduces t then interpolating H from trivariate images in $\mathbb{Z}_p[x_0, x_1, x_2]$ will likely reduce t further. The cost of Brown's GCD algorithm becomes $O(d^4)$ with $d = \max_{i=0}^{2}(\deg_{x_i} A, \deg_{x_i} B)$. For our benchmark problem $d^4 = 40^4 = 2,560,000$ which exceeds the cost of evaluation so there would need to be a significant reduction in t to see an overall gain.

Let us write

$$H = \sum_{i=0}^{d_0} h_i(x_1, \ldots, x_n)x_0^i = \sum_{i=0}^{d_0}\sum_{j=0}^{d_1} h_{ij}(x_2, \ldots x_n)x_0^i x_1^j$$

and define $t_1 = \max \#h_i$ and $t_2 = \max \#h_{ij}$. The ratio t_1/t_2 is reduction of the number of evaluation points needed by our algorithm. The maximum reduction in t occurs when the terms in H are distributed evenly over the coefficients of H in x_1, that is, then $t_1/t_2 = 1 + d_1 = 1 + \deg_{x_1} \Delta + \deg_{x_1} G$. For some very sparse inputs, there is no gain. For example,

for

$$H = x_0^d + x_1^d + x_2^d + \cdots + x_n^d + 1$$

we have $t_1 = n$ and $t_2 = n - 1$ and the gain is negligible.

If H has total degree D and H is dense then the number of terms in $h_i(x_1, \ldots, x_n)$ is $\binom{D-i+n}{n}$ which is a maximum for h_0 where $\#h_0 = \binom{D+n}{n}$. A conservative assumption is that $\#h_i$ is proportional to $\binom{n+D-i}{n}$ and similarly $\#h_{ij}$ is proportional to $\binom{n-1+D-(i+j)}{n-1}$. In this case, the reduction is a factor of

$$\frac{\#h_0}{\#h_{00}} = \binom{n+D}{n} / \binom{n-1+D}{n-1} = \frac{n+D}{n}.$$

For our benchmark problem where $n = 8$ and $D = 60$ this is $8.5 = \frac{68}{8}$.

Chapter 7

Benchmarks

We have implemented the GCD algorithm in Maple without any parallelism and have compared it with Maple's default algorithm, an implementation of a Zippel based algorithm. For most medium and large problems, our algorithm outperforms Maple's. For example, for input polynomials having 40 variables and 4000 terms, our algorithm is almost 20 times faster. We also attempted a parallel implementation (Threads package) of this algorithm in Maple but it was a flop. With parallelization in Maple, the GCD algorithm is even slower. I am told that this is because the prime is too large and the memory management in Maple has to create, simplify and garbage collect millions of multi-precision integers. Since the Maple's default algorithm is almost entirely coded in C and we coded our algorithm in Maple, it is an unfair comparison.

We have implemented algorithm PGCD for 31, 63 and 127 bit primes in Cilk C. For 127 bit primes we use the 128 bit signed integer type `__int128_t` supported by the gcc compiler. We parallelized evaluation (see Section 3.2) and we interpolate the coefficients $h_i(y)$ in parallel in step 12e.

The new algorithm requires $2t + \delta$ images (evaluation points) for the first prime and $t + 1$ images for the remaining primes. The additional image ($t + 1$ images instead of t) is used to check that the support of H obtained from the first prime is correct.

To assess how good our new algorithm is, we have compared it with the serial implementations of Zippel's algorithm in Maple 2016 and Magma V2.22. For Maple we are able to determine the time spent computing G modulo the first prime in Zippel's algorithm. It is typically over 99% of the total GCD time. The reason for this is that Zippel's algorithm requires $O(ndt)$ images for the first prime but only $t + 1$ images for the remaining primes.

We also timed Maple's implementation of Wang's EEZ-GCD algorithm from [Wang, 1980, 1978]. It was much slower than Zippel's algorithm on these inputs so we have not included timings for it. Note, older versions of Maple and Magma both used the EEZ-GCD algorithm for multivariate polynomial GCD computation.

All timings were made on the gaby server in the CECM at Simon Fraser University. This machine has two Intel Xeon E-2660 8 core CPUs running at 3.0 GHz on one core and 2.2 GHz on 8 cores. Thus the maximum parallel speedup is a factor of $16 \times 2.2/3.0 = 11.7$.

7.1 Benchmark 1

For our first benchmark (see Table 7.1) we created polynomials $G, \bar{A}$ and $\bar{B}$ in 6 variables ($n = 5$) and 9 variables ($n = 8$) of degree at most d in each variable. We generated $100d$ terms for G and 100 terms for $\bar{A}$ and $\bar{B}$. That is, we hold t approximately fixed to test the dependence of the algorithms on d.

The integer coefficients of $G, \bar{A}, \bar{B}$ were generated at random from $[0, 2^{31} - 1]$. The monomials in $G, \bar{A}$ and $\bar{B}$ were generated using random exponents from $[0, d - 1]$ for each variable. For G we included monomials $1, x_0^d, x_1^d, \ldots, x_n^d$ so that G is monic in all variables and $\Gamma = 1$. Maple and Magma code for generating the input polynomials is given in the Appendix.

Our new algorithm used the 62 bit prime $p = 29 \times 2^{57} + 1$. Maple used the 32 bit prime $2^{32} - 5$ for the first image in Zippel's algorithm.

			New GCD algorithm		Zippel's algorithm	
n	d	t	1 core (eval)	16 cores	Maple	Magma
5	5	110	0.29s (64%)	0.074s (3.9x)	3.57s	0.60s
5	10	114	0.62s (68%)	0.091s (6.8x)	48.04s	6.92s
5	20	122	1.32s (69%)	0.155s (8.5x)	185.70s	296.06s
5	50	121	3.48s (69%)	0.326s (10.7x)	1525.80s	$> 10^5 s$
5	100	123	7.08s (69%)	0.657s (10.8x)	6018.23s	NA
5	200	125	14.64s (71%)	1.287s (11.4x)	NA	NA
5	500	135	38.79s (71%)	3.397s (11.4x)	NA	NA
8	5	89	0.27s (61%)	0.065s (4.2x)	32.47s	2.28s
8	10	110	0.63s (65%)	0.098s (6.4x)	138.41s	7.33s
8	20	114	1.35s (66%)	0.163s (8.3x)	664.33s	78.77s
8	50	113	3.52s (66%)	0.336s (10.5x)	6390.22s	800.15s
8	100	121	7.43s (68%)	0.645s (11.5x)	NA	9124.73s

Table 7.1: Real times (seconds) for GCD problems.

In Table 7.1 column d is the maximum degree of the terms of $G, \bar{A}, \bar{B}$ in each variable, column t is the maximum number of terms of the coefficients of G. Timings are shown in seconds for the new algorithm for 1 core and 16 cores. For 1 core we show the %age of the time spent evaluating the inputs, that is computing $K(A)(x_0, \alpha^j)$ and $K(B)(x_0, \alpha^j)$ for $j = 1, 2, \ldots, T$. The parallel speedup on 16 cores is shown in parentheses.

Table 7.1 shows that most of the time in the new algorithm is in evaluation. It shows a parallel speedup approaching the maximum of 11.5 on this machine. There was a parallel bottleneck in how we computed the $c(z)$ polynomials that limited parallel speedup to 10 on

these benchmarks (not shown in Table 7.1). For N cores, after generating a new batch of N images we used the Euclidean algorithm for Step 12b which is quadratic in the number of images j computed so far. To address this we now use an incremental version of the Berlekamp-Massey algorithm which is $O(Nj)$.

7.2 Benchmark 2

Our second benchmark (see Table 7.2) is for 9 variables where the degree of $G, \bar{A}, \bar{B}$ is at most 20 in each variable. The terms are generated at random as before but restricted to have total degree at most 60. The row with $\#G = 10^4$ and $\#A = 10^6$ is our benchmark problem mentioned in Chapter 1. We show two sets of timings for our new algorithm. The first set is for projecting down to univariate image GCDs in $\mathbb{Z}_p[x_0]$ and the second set it for bivariate GCDs and consequently the values of t are different.

The timings for the new algorithm are for the first prime only. Although one prime is sufficient for these problems to recover H, that is, no Chinese remaindering is needed, our algorithm uses an additional 63 bit prime to verify $H \mod p_1 = H$. The time for the second prime is always less than 50% of the time for the first prime because it needs only $t+1$ points instead of $2t+\delta$ points and it does not need to compute degree bounds.

For $\#G = 10^3$, $\#A = 10^5$, the time of 497.2s breaks down as follows. 38.2s was spent in computing degree bounds for G, 451.2s was spent in evaluation, of which 43.2s was spent computing the powers. Using the support of H from this first prime it took 220.9s to compute H modulo a second prime.

Table 7.2 shows again that most of the time in the new algorithm is in evaluation. This is also true of Zippel's algorithm and hence of Maple and Magma too. Because Maple uses random evaluation points, and not a power sequence, the cost of each evaluation in Maple is $O(n(\#A+\#B))$ multiplications instead of $\#A+\#B$ evaluations for the new algorithm. Also, Maple is using `% p` to divide in C which generates a hardware division instruction which is much more expensive than a hardware multiplication instruction. For the new algorithm, we are using Roman Pearce's implementation of Möller and Granlund [Möller and Granlund, 2011] which reduces division by p to two multiplications plus other cheap operations. Magma is doing something similar. It is using floating point primes (25 bits) so that it can multiply modulo p using floating point multiplications. This is one reason why Maple is slower than Magma.

In comparing the new algorithm with Maple's implementation of Zippel's algorithm, for $n = 8, d = 50$ in Table 7.1 we achieve a speedup of a factor of $1815 = 6390.22/3.52$ on 1 core. Since Zippel's algorithm uses $O(dt)$ points and our Ben-Or/Tiwari algorithm uses $2t + O(1)$ points, we get a factor of $O(d)$ speedup because of this.

		New GCD: univariate images				New GCD: bivariate images				Zippel's algorithm	
$\#G$	$\#A$	t	1 core	(eval)	16 cores		t	1 core	(eval)	16 cores		Maple	Magma
10^2	10^5	13	0.14s	(62%)	0.043	3.3x	4	0.156s	(65%)	0.032s	2.8x	51.8s	10.92s
10^3	10^5	113	0.59s	(66%)	0.100s	5.9x	13	0.306s	(55%)	0.066s	4.6x	210.9s	60.24s
10^4	10^5	1197	7.32s	(48%)	1.022s	7.2x	118	2.299s	(36%)	0.224s	10.3x	7003.4s	10.84s
10^2	10^6	13	1.36s	(70%)	0.167s	8.2x	4	1.024s	(60%)	0.164s	6.2x	797.4s	45.08s
10^3	10^6	130	5.70s	(90%)	0.520s	11.0x	14	1.713s	(69%)	0.228s	7.1x	2135.9s	207.63s
10^4	10^6	1198	48.17s	(87%)	4.673s	10.3x	122	7.614s	(75%)	0.685s	11.1x	22111.6s	1611.46s
10^5	10^6	11872	466.09s	(82%)	45.83s	10.2x	1115	80.04s	(57%)	6.079s	13.2x	NA	876.89s
10^2	10^7	11	12.37s	(67%)	1.46s	8.5x	3	10.69s	(58%)	1.63s	6.6x	NA	354.90s
10^3	10^7	122	47.72s	(91%)	4.470s	10.7x	16	15.76s	(71%)	2.09s	7.5x	NA	1553.91s
10^4	10^7	1212	429.61s	(98%)	37.72s	11.4x	122	57.23s	(90%)	5.10s	11.2x	NA	8334.93s
10^5	10^7	11867	3705.4s	(98%)	311.6s	11.9x	1114	438.87s	(90%)	34.4s	12.8x	NA	72341.0s
10^6	10^7	117508	47568.s	(90%)	3835.9s	12.4x	11002	4794.5s	(83%)	346.1s	13.8x	NA	NA
10^2	10^8	12	129.26s	(69%)	15.86s	8.2x	4	101.8s	(60%)	16.88s	6.0x	NA	NA
10^3	10^8	121	522.14s	(92%)	49.17s	10.6x	17	150.0s	(73%)	23.25s	6.4x	NA	NA
10^4	10^8	1184	4295.0s	(99%)	412.69s	10.4x	121	555.5s	(89%)	78.76s	7.0x	NA	NA
10^5	10^8	11869	43551.s	(99%)	3804.93s	11.4x	1162	4417.7s	(98%)	626.19s	7.0x	NA	NA

Table 7.2: Timings (seconds) for 9 variable GCDs

Chapter 8

Computing polynomial GCD over number fields

In this chapter we present a "simplified" modular GCD algorithm which computes multivariate polynomial GCDs over an algebraic number field. This modular algorithm is similar to the simplified algorithm in Chapter 4 but has the capacity to handle zero divisors and reconstruct rational coefficients of the target GCD. The presence of zero divisors in the coefficient ring significantly complicates the analysis because a finite ring has less structure than a finite field and we lose many useful properties. Also we are forced to use the monic subresultant GCD algorithm (see Figure 8.2) instead of the monic Euclidean algorithm (see Figure 8.1) at the base case (univariate case) in $\mathbb{Z}_p(\alpha)[x]$ to simplify the analysis of zero divisors. We first review some results and define some notation for later use.

8.1 Some results and notation

We follow the definitions in van Hoeij and Monagan [van Hoeij and Monagan, 2002] but focus on the simple extension $\mathbb{Q}(\alpha)$ where $m(z) \in \mathbb{Q}[z]$ is the monic minimal polynomial of the algebraic number α. $\mathbb{Q}(\alpha)$ can also be expressed as $\mathbb{Q}[z]/\langle m(z)\rangle$ hence in this chapter α and z are interchangeable. Let $d_m = \deg_z m(z)$ where $d_m \geq 2$. If we consider $\mathbb{Q}(\alpha)$ as a vector space over $\mathbb{Q}$, then $\mathbb{Q}(\alpha)$ has dimension d_m over $\mathbb{Q}$ and a basis for $\mathbb{Q}(\alpha)$ is $M = \{\alpha^i \mid 0 \leq i < d\}$.

Definition 8.1. Let R be the set of all $\mathbb{Z}$-linear combinations of elements in M and $f \in \mathbb{Q}(\alpha)[x_0, x_1, \ldots, x_n]$. The *denominator* $den(f)$ of f is the smallest positive integer such that $den(f)f \in R[x_0, x_1, \ldots, x_n]$.

If $f \in \mathbb{Q}[x_0, x_1, \ldots, x_n]$, $den(f)$ is the least common multiple of denominators of coefficients of monomials in f.

Example 8.1. *Let* $f = \frac{2}{3}x + \frac{1}{4}$. *Then* $den(f) = 12$. *Let* $f = (\frac{2}{3}\alpha^2 + 2\alpha)x^2 + \frac{4}{5}\alpha x$. *Then* $den(f) = 15$.

Definition 8.2. Let R be a ring and $f \in R[x_0, x_1, \ldots, x_n]$. The leading coefficient of f with respect to $x_0, x_1, \ldots, x_n$ in some monomial order is denoted by $lc(f)$. If $lc(f) = 1$ then f is *monic*. Let x_0 be the main variable. Then the leading coefficient of f with respect to x_0 is denoted by $LC(f)$.

Example 8.2. *Let $f = \sqrt{2}z^2y^2x^3 + z^3y^4x + 1$ and we consider the pure lexicographical order with $x > y > z$. Then $lc(f) = \sqrt{2}$. If x is taken as the main variable, then $LC(f) = \sqrt{2}z^2y^2$.*

In this chapter we use the pure lexicographical order with $x_0 > x_1 > \cdots > x_n$ on monomials in $\mathbb{Q}(\alpha)[x_0, x_1, \ldots, x_n]$. Suppose $f \in \mathbb{Q}(\alpha)[x_0, x_1, \ldots, x_n]$. With a Kronecker substitution $K_r : \mathbb{Q}(\alpha)[x_0, x_1, \ldots, x_n] \to \mathbb{Q}(\alpha)[x, y]$ where $x_0 \to x$, $LC(K_r(f))$ denotes the leading coefficient of $K_r(f)$ with respect to the main variable x.

Definition 8.3. The *primitive associate* of f is $\tilde{f} = den(g)g$ where $g = monic(f) = lc(f)^{-1}f$. The *semi-associate* of f is cf where c is the smallest positive rational number to make $den(cf) = 1$.

Remark 8.1. r can be determined as follows. We compute the denominator of f, that is to compute the integer least common multiple (LCM) r_d of the denominators of coefficients in f. Then we compute the integer GCD r_n of the numerators of coefficients in f and set $r = \frac{r_d}{r_n}$. We also note that $\gcd(r_d, r_n) = 1$. If not, suppose $\gcd(r_d, r_n) = r_c$ and r_{cf} is a prime factor of r_c. Since r_d is the LCM, r_{cf} must divide the denominator of some coefficient, say C. Since r_n is the GCD, r_n must divide the numerator of every coefficient hence divides the numerator of C. Therefore the rational number C is not in the reduced form and the GCD of the numerator and the denominator in C is not 1. But we assume that input polynomials have reduced coefficients.

In practice if $lc(f)$ is a large element of $\mathbb{Q}(\alpha)$, then the rational coefficient of $lc(f)^{-1}$ can be about d_m times larger than the coefficients of $lc(f)$ and therefore $lc(f)^{-1}$ could be expensive to get. Therefore we preprocess all input polynomials by computing their semi-associate representations.

Example 8.3. *If $f = 4x - \frac{2}{3}$ then $c = \frac{3}{2}$ and $\tilde{f} = \check{f} = 6x - 1$. If α is the root of the minimal polynomial $m = z^4 + z^3 + 2$ and $f = -\alpha x + 1$, then $\check{f} = f$ but $monic(f) = x + \frac{\alpha^3+\alpha^2}{2}$ and $\tilde{f} = 2x + \alpha^3 + \alpha^2$.*

Any $\beta \in \mathbb{Q}(\alpha)$ can be expressed as the polynomial $\beta = \sum_{i=0}^{d_m-1} c_i\alpha^i \in \mathbb{Q}(\alpha)$ where $c_i \in \mathbb{Q}$. In actual computations β is regarded as the polynomial $\beta = \sum_{i=0}^{d_m-1} c_i z^i \in \mathbb{Q}[z]/\langle m(z)\rangle$. In our modular algorithm, the prime p used must be coprime to the denominators $den(A)$, $den(B)$, $den(m(z))$ of the inputs since otherwise the inputs modulo p do not exist. In general $\mathbb{Q}(\alpha) \mod p = \mathbb{Z}_p(\alpha) \cong \mathbb{Z}_p[z]/\langle m(z)\rangle$ is a finite ring instead of a finite field because $m(z) \mod p$ could be reducible and therefore every non-zero element in $\mathbb{Z}_p(\alpha)$ is either a unit or a zero divisor.

In this chapter, let $A, B \in \mathbb{Q}(\alpha)[x_0, x_1, \ldots, x_n]$ and $m(z)$ be input polynomials and $d = \max\{\max\{\deg_{x_i} A, \deg_{x_i} B\}\}$ for $0 \le i \le n$. d bounds the degree of every variable in A and B. We also assume that A and B are primitive polynomials with respect to the main variable x_0. Otherwise we remove the contents by recursively calling our polynomial GCD algorithm. Let $\check{A}$, $\check{B}$ and $\check{m}(z)$ be the semi-associates of A, B and $m(z)$ respectively. We mention that our GCD algorithm always computes the monic GCD. Let G be the monic GCD of $\check{A}$ and $\check{B}$. Then $G = \gcd(\check{A}, \check{B})$ because G is monic ($lc(G) = 1$). Let $\bar{A} = G/A, \bar{B} = G/B$ be the cofactors of A, B respectively. Let $D_m = den(m(z))$ which is also the leading coefficient of $\check{m}(z)$. We note that $\check{A}, \check{B}, \in \mathbb{Z}(\alpha)[x_0, x_1, \ldots, x_n]$, $G \in \mathbb{Q}(\alpha)[x_0, x_1, \ldots, x_n]$ and $d_m = \deg_z m(z) = \deg_z \check{m}(z)$. Let $\bar{\check{A}} = \check{A}/G$ and $\bar{\check{B}} = \check{B}/G$. We want to mention that $\bar{\check{A}} \neq \check{\bar{A}}$ and $\bar{\check{B}} \neq \check{\bar{B}}$ in general.

Example 8.4. *Let $m = z^2 + 2$ be the minimal polynomial for α. $A = (x + \frac{1}{5}\alpha)(\frac{2}{3}\alpha x^2 + 1)$ and $B = (x + \frac{1}{5}\alpha)(\frac{1}{5}\alpha x + \frac{3}{2})$. $G = \gcd(A, B) = x + \frac{1}{5}\alpha$. $\bar{A} = \frac{2}{3}\alpha x^2 + 1$ and $\bar{B} = \frac{1}{5}\alpha x + \frac{3}{2}$. Hence $\check{\bar{A}} = 2\alpha x^2 + 3$ and $\check{\bar{B}} = 2\alpha x + 15$. On the other hand, $\check{A} = 10\alpha x^3 - 4x^2 + 15x + 3\alpha$ and $\check{B} = 10\alpha x^2 + 71x + 15\alpha$. $\bar{\check{A}} = 10\alpha x^2 + 15$ and $\bar{\check{B}} = 10\alpha x^2 + 75$.*

Let $\Gamma = \gcd(LC(\check{A}), LC(\check{B}))$, $\Delta = \Gamma/LC(G)$ and $H = \Delta G$. Recall that $LC(\check{A})$ and $LC(\check{B})$ are the leading coefficients of $\check{A}$ and $\check{B}$ respectively and $\Gamma \in \mathbb{Q}(\alpha)[x_1, \ldots, x_n]$. Our algorithm computes H modulo a sequence of primes $p_1, p_2, \ldots$ As usual, not all primes can be used. It is clear that we can not use any prime p for which a denominator of inputs A or B vanishes modulo p or $D_m \mod p = 0$. If $\check{m}(z) \mod p$ is irreducible, then $\mathbb{Z}_p/\langle \check{m}(z) \rangle$ is still a field. All results for the GCD algorithm over finite fields (PGCD) hold here. Unfortunately it is possible that $\check{m}(z) \mod p$ is reducible for all primes. For example, $\check{m}(z) = z^4 + 1$. In general $\mathbb{Z}_p[z]/\langle \check{m}(z) \rangle$ is a finite ring with zero divisors. Let $f \in \mathbb{Q}(\alpha)[x_0, x_1, \ldots, x_n]$. We apply a Kronecker substitution K_r to reduce the number of variables where $K_r : \mathbb{Q}(\alpha)[x_0, x_1, \ldots, x_n] \to \mathbb{Q}(\alpha)[x, y]$ and $K_r(f) = f(x, y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots, r_{n-1}})$ for some $r_i > 0$. K_r can be bad or unlucky depending on the choice of r_i. After the Kronecker substitution, x is taken as the main variable. Our GCD algorithm may fail to determine $\gcd(K_r(\check{A}), K_r(\check{B})) \mod p$ even when if $\gcd(K_r(\check{A}), K_r(\check{B})) \mod p$ exists because the use of some evaluation points or primes could lead us to invert a zero divisor. Handling these bad cases is the new challenge in our GCD algorithm in this chapter.

Lemma 8.1 (Generalized Schwarz-Zippel Lemma)**.** *Let R be a commutative ring containing an integral domain D and let $f \in R[x_1, \ldots, x_n]$ be a nonzero polynomial with total degree $d \ge 0$. Let S be a finite subset of D. If β is chosen uniformly at random from S^n then $Prob[f(\beta) = 0] \le \frac{d}{|S|}$. Hence if $E = \{\beta | f(\beta) = 0\}$ then $|E| \le d|S|^{n-1}$.*

See [Arvind and Mukhopadhyay, 2007, Proposition 2, Lemma 10] for a proof.

In field theory, the *norm* maps elements of a larger field to a subfield. For example, we can map an element of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ to $\mathbb{Q}$ which is particularly useful in algebraic factorization.

Definition 8.4. Let K be a field and L be a finite extension of K. For any $\alpha \in L$ with minimal polynomial m over K, let $\alpha_1 = \alpha, \alpha_2, \cdots, \alpha_k$ be the conjugates of α. The norm of α

$$Norm(\alpha) = (\prod_{i=1}^{n} \alpha_i)^{[L:K(\alpha)]}.$$

For the simple extension $\mathbb{Q}(\alpha)$ from $\mathbb{Q}$, $Norm(\alpha) = \prod_{i=1}^{n} \alpha_i$ because $[\mathbb{Q}(\alpha) : \mathbb{Q}(\alpha)] = 1$. Similarly, for a polynomial $f(x_1, \ldots, x_k, \alpha) \in \mathbb{Q}(\alpha)[x_1, \ldots, x_k]$ where α is algebraic over $\mathbb{Q}$ with the minimal polynomial $m(z) \in \mathbb{Q}[z]$, we define

$$Norm(f) = \prod_{i=1}^{n} f(x_1, \ldots, x_k, \alpha_i).$$

Example 8.5. *Let $m(z) = z^2 - 2$ and $f = x + \sqrt{2} + 1$. Since the roots of $m(z)$ are $\sqrt{2}$ and $-\sqrt{2}$, $Norm(f) = (x + \sqrt{2} + 1)(x - \sqrt{2} + 1) = x^2 + 2x - 1$.*

We recall that the resultant $\mathrm{res}_z(m(z), r(z))$ is the determinant of the Sylvester's matrix of $m(z)$ and $r(z)$, see Equation (2.1). The following useful lemma can be found in [Geddes et al., 1992, Section 8.8].

Lemma 8.2. *Let $\beta \in \mathbb{Q}(\alpha)$ and $m(z) \in \mathbb{Q}[z]$ be the minimal polynomial of α over $\mathbb{Q}$. Suppose $\beta = r(\alpha)$ where $r \in \mathbb{Q}[z]$. Then $Norm(\beta) = \mathrm{res}_z(m(z), r(z))$.*

Proof. Let $\alpha_1 = \alpha, \alpha_2, \ldots, \alpha_k$ be conjugates of α. By definitions of the norm of a polynomial and the polynomial resultant, we have

$$Norm(\beta) = Norm(r(\alpha)) = \prod_{i=1}^{k} r(\alpha_i) = \mathrm{res}_z(m(z), r(z)).$$

□

Corollary 8.1. *Let $f \in \mathbb{Q}(\alpha)[x_1, \ldots, x_k]$ and $m(z) \in \mathbb{Q}[z]$ be the minimal polynomial of α. Then $Norm(f) = \mathrm{res}_z(m(z), f(x_1, x_2, \ldots, x_n, z))$.*

Proof. The proof is similar to the proof of the previous lemma. □

Lemma 8.3. *Let α be algebraic over $\mathbb{Q}$, p be a prime and $m(z)$ be the minimal polynomial of α over $\mathbb{Q}$. Assume $\beta \in \mathbb{Q}(\alpha)$, $\beta \mod p$ exists, $\beta \mod p \neq 0$ and $m(z) \mod p$ exists. If p divides $Norm(\beta)$, then $\beta \mod p$ is a zero divisor.*

Proof. Let $\beta = r(\alpha) = \sum_{i=0}^{n} b_i \alpha^i$ where $b_i \in \mathbb{Q}$. By Lemma 8.2 we have $Norm(\beta) = \mathrm{res}_z(m(z), r(z))$. Let ϕ_p be the modular homomorphism. Since $m(z)$ is monic and $r(z) \neq 0 \mod p$, by Lemma 2.4 we have

$$0 = \phi_p(\mathrm{res}_z(m(z), r(z))) \Longleftrightarrow \deg_z \gcd(\phi_p(m(z)), \phi_p(r(z))) > 0.$$

Hence $r(z)$ is not invertible in $\mathbb{Z}_p[z]/\langle m(z)\rangle$. So β is a zero divisor. □

Example 8.6. *Let $m(z) = z^3 - 2$ be the minimal polynomial of α and $\beta = \alpha + 2$. Since $Norm(\alpha + 2) = \text{res}_z(m(z), z + 2) = 10$, $\beta \mod p$ is a zero divisor if $p = 2$ or $p = 5$. If $p = 2$, then $\gcd(z + 2, z^3 - 2) \mod 2 = z$. If $p = 5$, $\gcd(z + 2, z^3 - 2) \mod 5 = z + 2$. In both cases, $\alpha + 2$ is not invertible.*

Definition 8.5. Let $f = \sum_{i=1}^{t} \frac{a_i}{b_i} M_i$ where $a_i, b_i \in \mathbb{Z} \setminus \{0\}$, $t \geq 1$ and the M_i are monomials in $x_1, x_2, \ldots, x_n$. If $b_i = 1$ for $1 \leq i \leq t$, then f is a polynomial with integer coefficients and we define $\|f\| = \max_{1 \leq i \leq t}\{|a_i|\}$ as the *height* of f. If $b_i \neq 1$ for some i then we define $|f|_{max} = \max_{1 \leq i \leq t}\{|\frac{a_i}{b_i}|\}$ and $|f|_{nmax} = \max_{1 \leq i \leq t}\{|a_i|\}$. We call $|f|_{nmax}$ as the *numerator magnitude* of f.

In the above definition, we note that if $b_i = 1$ for $1 \leq i \leq t$, then $|f|_{max} = |f|_{nmax} = \|f\|$. We also observe that $|f|_{nmax} \leq \|den(f)f\|$ which will be used several times in our analysis.

Lemma 8.4. *Let $A = \sum_{i=1}^{t_A} a_i u_i$ and $B = \sum_{j=1}^{t_B} b_j v_j$ where $t_A, t_B \geq 1$, $a_i, b_j \in \mathbb{Z}$, u_i, v_j are monomials in variables $x_1, x_2, \ldots, x_n$ and $u_{i_1} \neq u_{i_2}$ for $i_1 \neq i_2$, $v_{j_1} \neq v_{j_2}$ for $j_1 \neq j_2$. Then $\|AB\| \leq \min(t_A, t_B)\|A\|\|B\|$.*

Proof. First we note that the largest magnitude of the coefficients in the expanded form of AB is $\|A\|\|B\|$ if we don't add terms with identical monomials. Then we have to show if we expand AB term by term *without* adding coefficients, AB has at most $\min(t_A, t_B)$ terms with identical monomials. We assume that $t_A = t_B$, that is $t_A = t_B = \min(t_A, t_B)$. Since all v_j are distinct, all monomials in the polynomial $u_i B$ are distinct. Therefore there are at most t_A terms with identical monomials in the expansion of $\sum_{i=1}^{t_A} u_i B$. Now we assume that $t_A > t_B$, that is $t_B = \min(t_A, t_B)$. Since all u_i are distinct, all monomials in the polynomial $v_j A$ are distinct hence there are at most t_B terms with identical monomials in $\sum_{j=1}^{t_B} v_j A$. We can also count the number of terms with identical monomials by considering $u_i B$. Since all v_j are distinct, all monomials in the polynomial $u_i B$ are distinct, but we cannot conclude that the polynomial $\sum_{i=1}^{t_A} u_i B$ has at most t_A terms with identical monomials. If we suppose there are $t_B + 1$ terms with identical monomials in $\sum_{i=1}^{t_A} u_i B$ then there exists $u_{i_1} v_j = u_{i_2} v_j$ for $i_1 \neq i_2$. This implies $u_{i_1} = u_{i_2}$ for $i_1 \neq i_2$ which is a contradiction since all u_i are distinct. Therefore there are still at most t_B terms with identical monomials. If $t_A < t_B$, a similar argument shows $t_A = \min(t_A, t_B)$. The result follows. □

Lemma 8.5. *Let $a(x)$ and $b(x) \in \mathbb{Z}[x]$ with $\deg_x a = n, \deg_x b = m, LC(a) = a_n$ and $LC(b) = b_m$. Suppose $n \geq m$, q_0 and r_0 are the quotient and remainder of $a(x) \div b(x)$. Then we have*

$$|q_0|_{max} \leq \|a\|(1 + \|b\|/|b_m|)^{n-m}/|b_m|$$

$$|r_0|_{max} \leq \|a\|(1 + \|b\|/|b_m|)^{n-m+1}.$$

Proof. The proof follows the classic polynomial division algorithm.

Classic division algorithm
Input: $a(x), b(x) \in \mathbb{Z}[x]$ with $\deg_x a = n \geq 0 \geq \deg_x b = m \geq 0$. $LC(a) = a_n$ and $LC(b) = b_m$. $a(x)$ is dividend and $b(x)$ is divisor.
Output: the quotient q and remainder r in $\mathbb{Q}[x]$.
$r = a$
for i from $n - m$ to 0 by -1 do
  if $deg_x r = m + i$ then
    $q_i = LC(r)/b_m$
    $r = r - q_i x^i b$
  else
    $q_i = 0$
  end if
return r and $q = \sum_{i=0}^{n-m} q_i x^i$.

We bound the magnitudes of q and r at each iteration. For the first iteration, we have

$$|q_{n-m}|_{max} \leq \|a\|/|b_m|$$
$$|r_{n-m}|_{max} \leq \|a\| + |q_{n-m}|_{max}\|b\| \leq \|a\| + \|a\|\|b\|/|b_m| = \|a\|(1 + \|b\|/|b_m|).$$

For the second iteration, we have

$$\begin{aligned} |q_{n-m-1}|_{max} &\leq \|a\|(1 + \|b\|/|b_m|)/|b_m| \\ |r_{n-m-1}|_{max} &\leq \|r_{n-m}\| + |q_{n-m-1}|_{max}\|b\| \\ &\leq \|a\|(1 + \|b\|/|b_m|) + \|a\|(1 + \|b\|/|b_m|)\|b\|/|b_m| \\ &= \|a\|(1 + \|b\|/|b_m|)^2 \end{aligned}$$

We repeat this process and obtain

$$|q_0|_{max} \leq \|a\|(1 + \|b\|/|b_m|)^{n-m}/b_m \text{ and } |r_0|_{max} \leq \|a\|(1 + \|b\|/|b_m|)^{n-m+1}.$$

□

The bound derived in Lemma 8.5 for $|r_0|_{max} \in \mathbb{Q}$ is an upper bound for the absolute value of every coefficient in r_0. But in most of our applications we want $|r_0|_{nmax} \in \mathbb{Z}$. For example, $\log_{pmin} |r_0|_{nmax}$ is the maximum number of primes such that $r_0 \mod p = 0$ where p is a prime and $p \geq pmin$. The fact is that it is not easy to get $|r_0|_{nmax}$ and therefore we try to compute an upper bound of $|r_0|_{nmax}$ instead. If we scale $a(x)$ by $LC(b(x))^{n-m+1}$ which

is the strategy used by pseudo polynomial division, then $r_0 \in \mathbb{Z}$ hence $|r_0|_{max} = |r_0|_{nmax}$ which is an upper bound for the $|r_0|_{nmax}$ without scaling.

Example 8.7. *Let* $a = 3x^4 + 2x^3 + 9x^2 + 8x + 9$ *and* $b = 10x^3 + 3x^2 + 8x + 6$. *The remainder of* a *divided by* b *is* $r_0 = \frac{627}{100}x^2 + \frac{133}{25}x + \frac{417}{50}$ *and the bound* $9(1 + \frac{10}{10})^{4-3+1} = 36$ *derived in Lemma 8.5 bounds* $|r_0|_{max} = \frac{417}{50} = 8.34$ *but not* $|r_0|_{nmax} = 627$. *If we scale* a *by* $LC(b)^{4-3+1} = 10^2$ *then* $r_0 = 627x^2 + 532x + 834$ *and the bound* $900(1 + \frac{10}{10})^{4-3+1} = 3600 > 834 = |r_0|_{nmax} = |r_0|_{max} > 627$.

Lemma 8.6. *[Lenstra, 1987] Let* $f \in \mathbb{Q}(\alpha)[x_1, \ldots, x_n]$ *and* $m(z) \in \mathbb{Z}[z]$ *be the monic minimal polynomial of the algebraic* ***integer*** α. *Let* $d_i = \deg_{x_i} f$ *where* $1 \leq i \leq n$ *and* $d_m = \deg_z m(z)$. *For any monic factor* g *of* f, *the rational coefficients of* g *are bounded by*

$$|g|_{max} \leq e^D d_m (d_m - 1)^{\frac{d_m - 1}{2}} \|m\|_2^{d_m - 1} |discr(m)|^{-\frac{1}{2}} \sum_{i=0}^{d_m - 1} \|m\|_2^i |f|_{max},$$

where $\|m\|_2$ *is the 2-norm of* $m(z)$, $D = \sum_{i=1}^{n} d_i$ *and* $e = 2.78$

Lemma 8.6 only applies to algebraic integers. In our case we deal with algebraic numbers and therefore a proper conversion is required if we want to use it. The factors described in Lemma 8.6 are monic. Hence in our application we may also need a proper scaling. We also note here that if $f(x) \in \mathbb{Z}(\alpha)[x_1, \ldots, x_n]$, the factors of $f(x)$ are in $\mathbb{Q}(\alpha)[x_1, \ldots, x_n]$ instead of $\mathbb{Z}(\alpha)[x_1, \ldots, x_n]$. For example Weinberger and Rothschild [Weinberger and Rothschild, 1976] constructed the following example.

Example 8.8. *For* $f(x) = x^3 - 3 \in \mathbb{Q}(\alpha)[x]$ *where* $m(z) = z^6 + 3z^5 + 6z^4 + z^3 - 3z^2 + 12z + 16$. *The factorization of* $f(x)$ *over* $\mathbb{Q}(\alpha)$ *is*

$$\begin{aligned} f(x) = & (-\frac{1}{12}z^5 - \frac{1}{12}z^4 - \frac{1}{6}z^3 + \frac{7}{12}z^2 - \frac{11}{12}z + x - \frac{4}{3}) \\ & \times (x + \frac{1}{6}z^5 + \frac{1}{3}z^4 + \frac{2}{3}z^3 - \frac{1}{6}z^2 + \frac{2}{3}z + \frac{7}{3}) \\ & \times (-\frac{1}{12}z^5 - \frac{1}{4}^4 - \frac{1}{2}z^3 - \frac{5}{12}z^2 + \frac{1}{4}z + x - 1). \end{aligned}$$

Remark 8.2. For the simple extension of $\mathbb{Q}$ by an algebraic integer, the denominator of any factor of $f(x_0, \ldots, x_n) \in \mathbb{Q}(\alpha)[x_0, \ldots, x_n]$ is well studied, see [Lenstra, 1987, Encarnacion, 1995, Langemyr and McCallum, 1989]. Let α be an algebraic number and $m(z)$ be its minimal polynomial. Let D be a positive integer such that $f \in \frac{1}{D}\mathbb{Z}(\alpha)[x_0, \ldots, x_n]$ and $discr(m(z))$ be the discriminant of $m(z)$. Then all monic factors of f are in

$$\frac{1}{discr(m)D}\mathbb{Z}(\alpha)[x_0, \ldots, x_n].$$

In our case, our inputs are pre-processed to be semi-associates hence $D = 1$. In fact, let d be the largest integer such that d^2 divides $discr(m)$, then all monic factors of f are in

$\frac{1}{dD}\mathbb{Z}(\alpha)[x_0,\ldots,x_n]$. See [Encarnacion, 1995] for more details. Obviously d is better than $discr(m)$, but d could be too difficult to compute. In our analysis we use $discr(m)$, not d, to scale factors of a polynomial to clear fractions because several lemmas we want to apply require fraction free polynomial inputs.

Next we review the radical prime which will be used later.

Definition 8.6. Let $f(x) \in \mathbb{Q}[x]$. $f(x)$ is *squarefree* if there is no polynomial $u(x) \in \mathbb{Q}[x]$ where $\deg_x u(x) > 0$ such that $u^2|f$.

Theorem 8.1. *[Geddes et al., 1992, Theorem 8.1] Let $f(x) \in \mathbb{Q}[x]$, $f(x)$ is squarefree if and only if $\gcd(f, f') = 1$ where f' denotes the derivative of f.*

Definition 8.7. Let R be a commutative ring and I be an ideal of R. Then $\sqrt{I} = \{a \in R | a^n \in I\}$ is called *the radical of I*.

Definition 8.8. Let R be a commutative ring and I be an ideal of R. For any $a \in R$ and any positive integer n, if $a^n \in I$ implies $a \in I$ then I is a *radical ideal* hence $\sqrt{I} = I$.

Example 8.9. *Let $m(z) \in \mathbb{Q}[z]$. If $m(z)$ is not squarefree, then $\langle m(z)\rangle$ is not radical. This is because $m(z) = g(z)^2 h(z)$ for some $h(z), g(z) \in \mathbb{Q}[z]$ and $\deg_z g > 0$. $(g(z)h(z))^2 = (g(z)^2 h(z))h(z) = m(z)h(z) \in \langle m(z)\rangle$ but $g(z)h(z) \notin \langle m(z)\rangle$.*

Lemma 8.7. *Let R be a commutative ring and I be an ideal of R. $\sqrt{I}$ is the intersection of prime ideals of R containing I. Hence $\sqrt{I}$ is an ideal.*

This is a popular result in algebra. See [Tauvel and Yu, 2005, Proposition 2.5.5] for a proof.

Lemma 8.8. *Let F be a field, $m(z) \in F[z]$ and $m(z) \neq 0$. $m(z)$ is squarefree if and only if the ideal $\langle m(z)\rangle$ is a radical ideal.*

Proof. We first assume that $m(z)$ is squarefree. Let $m_1 \cdot m_2 \cdot m_k$ be the monic irreducible factorization of $m(z)$. Then $\gcd(m_i, m_j) = 1$ if $i \neq j$. By the property of ideals, we have

$$\langle m(z)\rangle = \langle m_1(z) \cdots m_k(z)\rangle = \langle m_1(z)\rangle \cap \cdots \cap \langle m_k(z)\rangle.$$

$\langle m(z)\rangle$ is the intersection of prime ideals of $F[z]$ and each $\langle m_i(z)\rangle$ contains $\langle m(z)\rangle$ for $1 \leq i \leq k$. Hence by Lemma 8.7 $\langle m(z)\rangle$ is radical.

For the other direction we assume $m(z)$ is not squarefree. Then $m(z) = AC^k$ for some $A, C \in F[z]$, C is irreducible and $k > 1$. Since $(AC)^k = A^{k-1}(AC^k)$, $(AC)^k \in \langle m(z)\rangle$. But $AC \notin \langle AC^k\rangle = \langle m(z)\rangle$. According to Definition 8.8, $\langle m(z)\rangle$ is not radical and therefore $\langle m(z)\rangle$ is not a radical ideal. □

Definition 8.9. Let $\langle m(z)\rangle$ be a radical ideal where $m(z) \in \mathbb{Q}[z]$. A prime p for $\langle m(z)\rangle$ is called a *radical prime* if p does not divide the denominator of $m(z)$ and $\langle m(z) \mod p\rangle$ remains radical.

Lemma 8.9. *Let $m(z) \in \mathbb{Q}[z]$ and $m(z) \neq 0$. Suppose $\langle m(z) \rangle = \sqrt{\langle m(z) \rangle}$. Then all but finite many primes are radical for $\langle m(z) \rangle$.*

Proof. By Lemma 8.8, $m(z)$ is squarefree hence $\gcd(m(z), m'(z)) = 1$ over $\mathbb{Q}$. By the Euclidean algorithm there exist polynomials $s(z), t(z) \in \mathbb{Q}[z]$ such that $s(z)m(z)+t(z)m'(z) = 1$. We use any prime p which does not divide the denominators of $m(z), m'(z), s(z), t(z)$ so that $s(z)m(z) + t(z)m'(z) \equiv 1 \mod p$ holds. This implies $\gcd(m(z), m'(z)) = 1 \in \mathbb{Z}_p[x]$ hence $m(z) \mod p$ is squarefree. Lemma 8.8 implies that $\langle m(z) \mod p \rangle$ remains radical. Since there are only finitely many primes dividing the denominators of $m(z), m'(z), s(z), t(z)$, the result follows. □

Lemma 8.10. *Let $m(z) \in \mathbb{Q}[z]$ and $m(z) \neq 0$. Suppose $\langle m(z) \rangle = \sqrt{\langle m(z) \rangle}$. There are at most $\log_{pmin} \lceil (|m|_{nmax}^{\deg_z m} + (\deg_z m |m|_{nmax})^{\deg_z m)} \rceil$ primes p with $p > pmin$ which divide the denominator of $m(z)$ and are not radical.*

Proof. Suppose p does not divide the denominator of $m(z)$. If p is not a radical prime, then $\langle m(z) \rangle \mod p$ is not radical. Therefore according to Lemma 8.8 $\gcd(m(z), m'(z)) \neq 1 \mod p$ which implies $\mathrm{res}(m(z), m'(z)) = 0 \mod p$ by Lemma 2.4. Let $h = |m|_{nmax}$, then $|m'|_{nmax} \leq \deg_z m h$. $\mathrm{res}(m(z), m'(z))$ is the determinant of the Sylvester's matrix of $m(z)$ and $m'(z)$ and therefore the first $\deg_z m'$ columns consist of 0 and the coefficients of $m(z)$ and the last $\deg_z m$ columns consist of 0 and the coefficients of $m'(z)$. Hence the magnitude of the numerator of $\mathrm{res}(m(z), m'(z))$ is bounded by

$$|\mathrm{res}(m(z), m'(z))|_{nmax} \leq h^{\deg_z m'} + (\deg_z mh)^{\deg_z m}.$$

□

8.2 An example

In this section, we demonstrate our algorithm by an example. Since any non-zero element in a field is a unit, every non-zero pair of elements in $\mathbb{Q}(\alpha)$ are associates. Traditionally the polynomial GCD over a number field is defined to be the monic one. This simplified algorithm assumes that the number of non-zero terms in the target GCD is given, and therefore we cannot run into an infinite loop when we try to determine stable feedback polynomials by the Berlekamp-Massey algorithm. So we use $\Gamma = \gcd(LC(A), LC(B))$ as the scaling factor. Recall that this scaling method may introduce polynomial fractions. See Section 2.8.

Example 8.10. *Consider the GCD problem $G = \sqrt{-1}x+y$, $A = G(x+1)$ and $B = G(x+2)$. Since the GCD algorithm always computes the monic GCD over number fields, we have $g = \gcd(A, B) = x - \sqrt{-1}y$. We notice that g and G divide each other but g is the GCD because $lc(g) = 1$.*

We present an example to compute the bivariate polynomial GCD over a number field and therefore no Kronecker substitution is required. In fact, a Kronecker substitution works over any integral domain, and therefore all results about it in the GCD algorithm over integers also apply to the GCD algorithm over number fields in this chapter.

Example 8.11. *Consider the following GCD problem. Let α be an algebraic number with the minimal polynomial $m(z) = z^3 + \frac{1}{3}$,*

$$G = yx^2 + (-4y^3 - 83)\frac{\alpha^2}{6}x + (3y^3 + 54y)\frac{\alpha}{7}x,$$

$$\bar{A} = x + y + z \quad \textit{and} \quad \bar{B} = x + y + 2z.$$

Let x be the main variable, y be the variable to be interpolated, $A = G\bar{A}$ and $B = G\bar{B}$. We want to compute the GCD of A and B.

We note that A and B are primitive with respect to x, this can be checked if we expand $G\bar{A}$ and $G\bar{B}$.

We first convert A and B to $\check{A}$ and $\check{B}$ to eliminate the denominators and remove integer contents. The leading coefficients of $\check{A}$ and $\check{B}$ are both $126y$ with respect to x hence the scaling factor is $\Gamma = \gcd(LC(\check{A}), LC(\check{B})) = y$. Note that $\Gamma = y$ is monic, and therefore the scaled univariate images are monic too. We mention that it looks like the leading coefficients of $\check{A}$ and $\check{B}$ are $42y$ because the denominator of G is 42. But the true leading coefficients are $126y$. The extra factor 3 comes from the reduction of $G\bar{A}$ and $G\bar{B}$ by $m(z)$.

We use the prime $p = 101$ and pick the generator $\omega = 2$. The shift $s = 92$ is chosen uniformly at random from $\mathbb{Z}_p$. Therefore the evaluation points for y are $\omega^s, \omega^{s+1}, \omega^{s+2}, \ldots$. We run the first iteration to compute the first two univariate GCD images over $\mathbb{Z}_p(\alpha)$ by the monic Euclidean algorithm over $\mathbb{Z}_p(\alpha)$ then scale the results and obtain

$$\begin{aligned} g_0 &= \omega^{s+0} \gcd(\check{A}(x, \omega^{s+0}), \check{B}(x, \omega^{s+0})) = 58x^2 + 50\alpha^2 x + 43\alpha x, \\ g_1 &= \omega^{s+1} \gcd(\check{A}(x, \omega^{s+1}), \check{B}(x, \omega^{s+1})) = 15x^2 + 76\alpha^2 x + 80\alpha x. \end{aligned}$$

The coefficient of x^2 is known to be y, the scaling factor, hence we only need to recover the coefficients of $\alpha^2 x$ and αx. Running the Berlekamp-Massey algorithm on inputs $[50, 76]$ and $[43, 80]$ we obtain the feedback polynomials $1 + 51v$ and $1 + 78v$ respectively. We run a new iteration to compute the next two scaled univariate GCD images and obtain

$$\begin{aligned} g_2 &= \omega^{s+2} \gcd(\check{A}(x, \omega^{s+2}), \check{B}(x, \omega^{s+2})) = 30x^2 + 82\alpha^2 x + 89\alpha x, \\ g_3 &= \omega^{s+3} \gcd(\check{A}(x, \omega^{s+3}), \check{B}(x, \omega^{s+3})) = 60x^2 + 29\alpha^2 x + 23\alpha x. \end{aligned}$$

Running the Berlekamp-Massey algorithm on inputs $[50, 76, 82, 29]$ and $[43, 80, 89, 23]$ we obtain the feedback polynomials $8v^2 + 92v + 1$ and $32v^2 + 89v + 1$ respectively. The degrees of both feedback polynomials increased so we run the next iteration to compute the next two

scaled univariate GCD images.

$$g_4 = \omega^{s+4} \gcd(\check{A}(x, \omega^{s+4}), \check{B}(x, \omega^{s+4})) = 19x^2 + 9\alpha^2 x + 54\alpha x,$$
$$g_5 = \omega^{s+5} \gcd(\check{A}(x, \omega^{s+5}), \check{B}(x, \omega^{s+5})) = 38x^2 + 51\alpha^2 x + 13\alpha x.$$

Running the Berlekamp-Massey algorithm on $[50, 76, 82, 29, 9, 51]$ *and* $[43, 80, 89, 23, 54, 13]$ *we obtain the feedback polynomials* $8v^2 + 92v + 1$ *and* $32v^2 + 89v + 1$ *respectively. Both feedback polynomials are unchanged from the previous iteration. Hence we terminate the iteration and compute the roots of the reciprocals of both polynomials which are* $\{8, 1\}$ *and* $\{8, 4\}$*. Since* $\log_\omega 8 = 3, \log_\omega 4 = 2$ *and* $\log_\omega 1 = 0$*, the support for the coefficient of* $\alpha^2 x$ *is* $\{y^3, y^0\}$ *and the support for the coefficient of* αx *is* $\{y^3, y^2\}$*. With the roots we set two Vandermonde systems. For the coefficient of* $\alpha^2 x$*, we have*

$$\begin{bmatrix} 8^{92} & 1^{92} \\ 8^{93} & 1^{93} \end{bmatrix} \begin{bmatrix} c_{1,1} \\ c_{1,2} \end{bmatrix} = \begin{bmatrix} 50 \\ 76 \end{bmatrix}.$$

The solution to this system is $\{c_{1,1} = 33, c_{1,2} = 3\}$*. For the coefficient of* αx*, we have*

$$\begin{bmatrix} 8^{92} & 4^{92} \\ 8^{93} & 4^{93} \end{bmatrix} \begin{bmatrix} c_{2,1} \\ c_{2,2} \end{bmatrix} = \begin{bmatrix} 43 \\ 80 \end{bmatrix}.$$

The solution to this system is $\{c_{2,1} = 87, c_{2,2} = 51\}$*. By solving the shifted Vandermonde system we obtain the coefficients of all elements in both supports and get the GCD of* $\check{A}$ *and* $\check{B}$ *modulo* 101*:*

$$G_{101} = x^2 y + (33y^3 + 3)\alpha^2 x + (87y^3 + 51y^2)\alpha x.$$

Now we run the rational number reconstruction on all coefficients of G_{101} *and obtain*

$$K_{101} = yx^2 + (-\frac{2}{3}y^3 + 3)\alpha^2 x + (\frac{3}{7}y^3 + \frac{1}{2}y^2)\alpha x.$$

However K_{101} *does not divide* A *or* B *in* $\mathbb{Q}(\alpha)[x, y]$*. Hence it's not the correct GCD and we need more modular images. Note that* p *could be an unlucky prime which can be detected if more images are computed. For now, we assume the support of* K_{101} *is correct and we set up the assumed form*

$$g_{form} = ax^2 y + (by^3 + c)\alpha^2 x + (dy^3 + ey^2)\alpha x,$$

where a, b, c, d, e *are unknown constants. Since* $\Gamma = y$*,* $a = 1$*. We choose the prime* $p = 103$ *for the new image and pick the generator* $\omega = 5$*. The shift* $s = 5$ *is uniformly and randomly chosen from* $\mathbb{Z}_{103}$*. In* g_{form} *the number of terms in* $\alpha^2 x$ *and* αx *are both* 2 *hence we compute*

two scaled univariate GCD images

$$g_0 = \omega^{s+0} \gcd(\check{A}(x, \omega^{s+0}), \check{B}(x, \omega^{s+0})) = 35x^2 + 54\alpha^2 x + 15\alpha x,$$
$$g_1 = \omega^{s+1} \gcd(\check{A}(x, \omega^{s+1}), \check{B}(x, \omega^{s+1})) = 72x^2 + 88\alpha^2 x + 46\alpha x.$$

At the same time we evaluate g_{form} *and obtain*

$$g_{form}(x, \omega^{s+0}) = 35x^2 + (27b + c)\alpha^2 x + (27d + 92e)\alpha x,$$
$$g_{form}(x, \omega^{s+1}) = 72x^2 + (79b + c)\alpha^2 x + (79d + 34e)\alpha x.$$

We set two linear systems to solve b, c, d, e *by extracting the coefficients of equations* $g_0 = g_{form}(x, \omega^{s+0})$ *and* $g_1 = g_{form}(x, \omega^{s+1})$ *and obtain*

$$\begin{bmatrix} 27 & 1 \\ 79 & 1 \end{bmatrix} \begin{bmatrix} b \\ c \end{bmatrix} = \begin{bmatrix} 54 \\ 88 \end{bmatrix} \quad \textit{and} \quad \begin{bmatrix} 27 & 92 \\ 79 & 34 \end{bmatrix} \begin{bmatrix} d \\ e \end{bmatrix} = \begin{bmatrix} 15 \\ 46 \end{bmatrix}.$$

Both linear systems can be solved efficiently by the shifted Vandermonde system solving algorithm and the solutions are $\{b = 68, c = 72\}, \{d = 74, e = 96\}$. *Evaluating the assumed form at the solutions, we have*

$$G_{103} = yx^2 + (68y^3 + 72)\alpha^2 x + (74y^3 + 96y^2)\alpha x.$$

We note that both linear systems have solutions hence $p = 101$ *is likely not an unlucky prime. We can also use the technique used in our faster GCD algorithm over* $\mathbb{Z}$ *by evaluating* y *in* A, B, G *at random point* e *modulo* 103 *and checking if* $G_{103}(x, e) | \gcd(A(x, e), B(x, e))$. *Now we apply the Chinese remaindering to* G_{101} *and* G_{103} *and obtain*

$$K_2 = yx^2 + (3467y^3 + 1720)\alpha^2 x + (5945y^3 + 2980y^2)\alpha x \mod 101 \cdot 103.$$

We apply the rational reconstruction algorithm to all coefficients of K_2. *If we use Wang's rational reconstruction algorithm (use command* ***iratrecon(1720,101*103)*** *in Maple) as mentioned in Section 1.8, then the reconstruction is unable to determine the rational coefficient* $\frac{a}{b}$ *of* $\alpha^2 x$ *so that* $\frac{a}{b} = 1720 \mod 101 \cdot 103$. *This is because* $\sqrt{\frac{101 \cdot 103}{2}} \approx 72 < 83 = a$. *If we compute one more GCD image with the prime* 107 *then* $\sqrt{\frac{101 \cdot 103 \cdot 107}{2}} \approx 746 > 83 = a$ *and therefore Wang's algorithm can determine* $\frac{a}{b}$ *uniquely. If we use Monagan's maximal quotient rational reconstruction algorithm (use command* ***iratrecon(1720,101*103,maxquo,1)*** *in Maple), then* $\frac{a}{b} = \frac{83}{6} = 1720 \mod 101 \cdot 103$ *can be determined immediately. We run Monagan's algorithm to all coefficients and get*

$$G = yx^2 + (-\frac{2}{3}y^3 - \frac{83}{6})\alpha^2 x + (\frac{3}{7}y^3 + \frac{54}{7}y^2)\alpha x.$$

Since the content of G is $cont(G) = \gcd(y, (-\frac{2}{3}y^3 - \frac{83}{6})\alpha^2, (\frac{3}{7}y^3 + \frac{54}{7}y^2)\alpha) = 1$, the primitive part of G is G. Now we test if $G|A$ and $G|B$ in $\mathbb{Q}(\alpha)[x, y]$. Since this is the case, we conclude $G = \gcd(A, B)$.

The new challenge compared to the polynomial GCD algorithm over integers is the possible presence of zero divisors which is not demonstrated in Example 8.2. Any calculation involving the algebraic number inversion modulo a prime may fail due to zero divisors. Fortunately the number of zero divisors is finite as we will show later. If we can also detect and avoid bad and unlucky primes and bad and unlucky evaluation points, then rational number reconstruction will stabilize once the modulus $m = p_1 p_2 \cdots$ is large enough , and therefore the algorithm will eventually terminate and output the correct polynomial GCD.

8.3 Bad and unlucky Kronecker substitutions

In our new algorithm, we perform the Kronecker substitution over $\mathbb{Q}(\alpha)$ which is a field. The bad or unlucky Kronecker substitutions only depend on $\deg_{x_i} A$ and $\deg_{x_i} B$ for $0 \le i \le n$. Therefore the result for the simplified GCD algorithms over $\mathbb{Z}$ hold for $\mathbb{Q}(\alpha)$. Equation 4.2 in Chapter 4 states that if we use K_r with the sequence

$$[r_i = 2d^2 + 1 \ge (\deg_{x_i} A \deg_{x_0} B + \deg_{x_i} A \deg_{x_0} B) + 1 : 1 \le i \le n]$$

then the Kronecker substitution is always not bad and not unlucky. If a Kronecker substitution is not bad and not unlucky we call it *good*. From now on K_r is assumed to be the Kronecker substitution with the sequence $[r_i = 2d^2 + 1 : 1 \le i \le n]$. Hence K_r is always good in this chapter. By the choice of r_i we have $\deg_y K_r(A) < (2d^2 + 1)^n$ and $\deg_y K_r(B) < (2d^2 + 1)^n$. See Proposition 4.1 for a proof.

We mention again that the choice of r_i guarantees the map K_r is invertible for A and B, therefore $\#A = \#K_r(A)$ and $\#B = \#K_r(B)$. See Remark 4.1. In our simplified algorithm we use $\Gamma = \gcd(LC(A), LC(B)) = \Delta LC(G)$ as the scaling factor. Since $\Delta = \gcd(LC(\bar{A}), LC(\bar{B}))$, $\deg_{x_i} LC(\bar{A}) \le \deg_{x_i} \bar{A}$ for $1 \le i \le n$ which implies $\deg_{x_i} H = \deg_{x_i} \Delta G \le \deg_{x_i} A$. Similarly we have $\deg_{x_i} H = \deg_{x_i} \Delta G \le \deg_{x_i} B$. Therefore K_r is also invertible for H and G. Please note that our algorithm first computes H then determines G by removing the content of H with respect to the main variable x_0.

Remark 8.3. Converting the inputs A and B to $\check{A}$ and $\check{B}$ does not affect the Kronecker substitution which only depends on the degrees of variables in inputs.

8.4 Bad and unlucky primes

In this section, the Kronecker substitution is good and invertible for $A, B, \Delta G$ and G. Recall that $\check{A}$, $\check{B}$ and $\check{m}(z)$ are the semi-associates of A, B and $m(z)$ respectively.

Definition 8.10. Suppose K_r is a good Kronecker substitution, $g = \gcd(K_r(\check{A}), K_r(\check{B}))$, $K_r(\bar{\check{A}}) = K_r(\check{A})/g$ and $K_r(\bar{\check{B}}) = K_r(\check{B})/g$. For a prime p, if any of $LC(K_r(\check{A}))$, $LC(K_r(\check{B}))$ or the leading coefficient of $\check{m}(z)$ vanish modulo p, then p is a *bad prime*. Suppose p is not bad and the Euclidean algorithm successfully determines $\bar{g} = \gcd(\phi_p(K_r(\bar{\check{A}}), \phi_p(K_r(\bar{\check{B}})))$. Then p is *unlucky* if $\deg_x \bar{g} > 0$. If p is not bad and not unlucky, we call it *good*.

Example 8.12. *Consider the following GCD problem.*

$$K_r(A) = \frac{7}{6}x^3 + \frac{145}{234}\alpha x^2 y + \frac{1}{39}xy^2,$$
$$K_r(B) = \frac{7}{6}x^3 + \frac{145}{234}\alpha x^2 y + \frac{77}{18}x^2 + \frac{1}{39}xy^2 + \frac{11}{13}\alpha xy,$$

where α is an algebraic number with minimal polynomial $m(z) = z^2 - 1/3$. We have $g = \gcd(K_r(A), K_r(B)) = x^2 + \frac{18}{91}\alpha xy$, $K_r(\bar{A}) = K_r(A)/g = \frac{7}{6}x + \frac{7}{18}\alpha y$ and $K_r(\bar{B}) = K_r(B)/g = \frac{7}{6}x + \frac{7}{18}\alpha y + \frac{77}{18}$. The semi-associatives of $K_r(A)$ and $K_r(B)$ are

$$K_r(\check{A}) = 273x^3 + 145\alpha x^2 y + 6xy^2 \quad \textit{and}$$
$$K_r(\check{B}) = 273x^3 + 145\alpha x^2 y + 1001x^2 + 6xy^2 + 198\alpha xy.$$

Since $\check{m}(z) = 3z^2 - 1$, 3 is a bad prime. $LC(K_r(\check{A})) = LC(K_r(\check{B})) = 273 = 3 \cdot 7 \cdot 13$ hence 7 and 13 are bad primes too. $\gcd(K_r(\check{A}), K_r(\check{B})) \mod 11 = x^3 + 4x^2 y\alpha + 7xy^2$ hence 11 is an unlucky prime. Since $\operatorname{res}_z(K_r(\bar{A}), K_r(\bar{B})) = \frac{539}{324}y$ where $539 = 7^2 \cdot 11$, 11 is the only unlucky prime because 7 has already been ruled out as a bad prime.

Let $\check{A} = \sum_{i=0}^{d_A} a_i(\alpha, x_1, \ldots, x_n)x_0^i$, $\check{B} = \sum_{i=0}^{d_B} b_i(\alpha, x_1, \ldots, x_n)x_0^i$, $h > \max\{\|\check{A}\|, \|\check{B}\|\}$ for some constant $h \in \mathbb{Z}$ and $d \geq \max_{i=0}^{n}\{\max\{\deg_{x_i} \check{A}, \deg_{x_i} \check{B}\}\}$. Note, $d_A, d_B \leq d$. Let $m(z)$ be the minimal polynomial of the algebraic number α, $d_m = \deg_z m(z)$ and $D_m = den(m(z))$. In order to apply Lemma 8.6, we need to convert the algebraic number α to an algebraic integer $C\alpha$ for some constant C. Let $m(z) = z^{d_m} + c_{d_m-1}z^{d_m-1} + c_{d_m-2}z^{d_m-2} + \cdots + c_0$ where $c_i \in \mathbb{Q}$ for $1 \leq i \leq d_{m-1}$. We define

$$s(v) = v^{d_m} + D_m c_{d_m-1} v^{d_m-1} + \cdots + D_m^{d_m} c_0 = 0.$$

The polynomial $s(v)$ is monic, irreducible over $\mathbb{Q}$, and has integer coefficients. $s(v)$ has the root $D_m\alpha$ since $s(D_m\alpha) = D_m^{d_m} m(\alpha) = 0$. Therefore $D_m\alpha$ is an algebraic integer with minimal polynomial $s(v)$. Also $\deg_v s(v) = d_m$ and $C = D_m$. Since $D_m\alpha \in \mathbb{Q}(\alpha)$ and $\alpha = \frac{1}{D_m} D_m\alpha \in \mathbb{Q}(D_m\alpha)$, under the mapping $\alpha \mapsto D_m\alpha$ we have

$$\mathbb{Q}[z]/\langle m(z)\rangle \cong \mathbb{Q}(\alpha) \cong \mathbb{Q}(D_m\alpha) \cong \mathbb{Q}[v]/\langle s(v)\rangle,$$

We extend this isomorphism to polynomial rings so that

$$\mathbb{Q}(\alpha)[x_0,\ldots,x_n] \cong \mathbb{Q}(D_m\alpha)[x_0,\ldots,x_n].$$

As the consequence of the conversion, we also need to represent the coefficients of $\check{A}$ and $\check{B}$ in terms of $D_m\alpha$, that is we have to replace α by $\frac{1}{D_m}D_m\alpha$ in $\check{A}$ and $\check{B}$. The highest degree of z (representing α) in $\check{A}$ or $\check{B}$ is d_m-1. Therefore, after the conversion, the largest possible denominator in $\check{A}$ and $\check{B}$ is $D_m^{d_m-1}$. Let A_c and B_c denote the converted $\check{A}$ and $\check{B}$ respectively, that is $A_c, B_c \in \frac{1}{D_m^{d_m-1}}\mathbb{Z}(D_m\alpha)[x_0,x_1,\ldots,x_n]$. Then we use $\check{A}_c$ and $\check{B}_c$ to denote the semi-associatives of A_c and B_c respectively and $\check{A}_c, \check{B}_c \in \mathbb{Z}(D_m\alpha)[x_0,x_1,\ldots,x_n]$. Let $\mathrm{lc}(\check{A}_c)$ and $\mathrm{lc}(\check{B}_c)$ denote the leading coefficient of $\check{A}_c$ and $\check{B}_c$ with respect to all variables in lexicographic order $x_0 > x_1 > \cdots > x_n$. Since $\|\check{A}\| < h$ and $\|\check{B}\| < h$, $\|\check{A}_c\| < D_m^{d_m-1}h$ and $\|\check{B}_c\| < D_m^{d_m-1}h$. Let $discr(s) = (-1)^{\binom{d_m}{2}}\mathrm{res}_v(s(v), s'(v)) \in \mathbb{Z}$ be the discriminant of $s(v)$. By Remark 8.2, any *monic factor* of $\check{A}_c$ or $\check{B}_c$ is in $\frac{1}{discr(s)}\mathbb{Z}(D_m\alpha)[x_0,x_1,\ldots,x_n]$. We note that $discr(s)monic(\bar{\check{A}}_c)$ and $discr(s)monic(\bar{\check{B}}_c)$ have integer coefficients.

Let $h_c = \max(|A_c|_{nmax}, |B_c|_{nmax})$ and $d_c = \max(den(A_c), den(B_c))$. Then we have $\max(\|\check{A}_c\|, \|\check{B}_c\|) \le d_c h_c$. We first determine the maximum number of bad primes.

Theorem 8.2. *Let $\mathbf{B}_{bp} = d_c^2 h_c^2$. There are at most $\lceil \log_{pmin} B_{bp} \rceil$ bad primes p with $p \ge pmin$ for the inputs $\check{A}_c$ and $\check{B}_c$.*

Proof. After the algebraic integer conversion the minimal polynomial $s(v)$ is monic and it does not contribute to the number of bad primes. A bad prime p divides all coefficients of numerators of $LC(K_r(\check{A}_c))$ or $LC(K_r(\check{B}_c))$. Since K_r is assumed to be invertible for A_c and B_c, $\#K_r(\check{A}_c) = \#\check{A}_c$ and $\#K_r(\check{B}_c) = \#\check{B}_c$. Therefore no monomials are identical after the Kronecker substitution and therefore all coefficients remain the same. We have $\|LC(K_r(\check{A}_c))\| \le \|K_r(\check{A}_c)\| = \|\check{A}_c\| \le d_c h_c$ or $\|LC(K_r(\check{B}_c))\| \le \|K_r(\check{B}_c)\| = \|\check{B}_c\| \le d_c h_c$. Let $\mathbf{B_{bp}} = d_c^2 h_c^2$, the result follows. □

Theorem 8.3. *Let*

$$\mathbf{B}_{up} = (2d)^d((d_m)(d_m-1)(2d^2+1)^n E|discr(s)|d_c h_c)^{2d}(1+\|s(v)\|)^{(4d-1)(d_m-1)}$$

where $E = e^{(n+1)d}d_m(d_m-1)^{\frac{d_m-1}{2}}\|s(v)\|_2^{d_m-1}|discr(s)|^{-\frac{1}{2}\sum_{i=0}^{d_m-1}}\|s(v)\|_2^i$, $e = 2.78$ and $\|\cdot\|_2$ denotes the 2-norm. Then there are at most $\lceil \log_{pmin} B_{up} \rceil$ unlucky primes p with $p \ge pmin$ for the inputs A_c and B_c, provided that all those primes are not bad.

Remark 8.4. The length of $\mathbf{B}_{up}$ is polynomial in $d_m, n, d, \log_{pmin} h_c$.

Proof. We want to show that $\mathbf{B}_{up}$ is an upper bound for $|\mathrm{res}_x(K_r(\bar{\check{A}}_c), K_r(\bar{\check{B}}_c))|_{nmax}$. We first scale $monic(\bar{\check{A}}_c)$ and $monic(\bar{\check{B}}_c)$ by $discr(s)$ to get integer coefficients. By Lemma 8.6

we have

$$\|discr(s)monic(\bar{\check{A}}_c)\| \le E|discr(s)|\|\check{A}_c\| \quad \text{and} \quad \|discr(s)monic(\bar{\check{B}}_c)\| \le E|discr(s)|\|\check{B}_c\|,$$

where $E = e^{nd}d_m(d_m-1)^{\frac{d_m-1}{2}}\|s(v)\|_2^{d_m-1}|discr(s)|^{-\frac{1}{2}}\sum_{i=0}^{d_m-1}\|s(v)\|_2^i$ and $\|\cdot\|_2$ denotes the 2-norm.

In the above two equations we have $\|discr(s)monic(\bar{\check{A}}_c)\|$ and $\|discr(s)monic(\bar{\check{A}}_c)\|$. But we really want $\|discr(s)\bar{\check{A}}_c\|$ and $\|discr(s)\bar{\check{A}}_c\|$. Besides, our GCD algorithm computes the monic GCD, $\mathrm{lc}(\check{A}_c) \in \mathbb{Q}(D_m\alpha)$ and $\mathrm{lc}(\check{B}_c) \in \mathbb{Q}(D_m\alpha)$ both belong to the cofactors $\bar{\check{A}}_c$ and $\bar{\check{B}}_c$. Therefore we scale $monic(\bar{\check{A}}_c)$ and $monic(\bar{\check{B}}_c)$ by $\mathrm{lc}(\check{A}_c)$ and $\mathrm{lc}(\check{B}_c)$ respectively. The scaling operation will not introduce fractions because $D_m\alpha$ is an algebraic integer. $\mathrm{lc}(\check{A}_c)$ and $\mathrm{lc}(\check{B}_c)$ both have degree at most d_m-1 in v and $\|\mathrm{lc}(\check{A}_c)\| \le \|\check{A}_c\|$ and $\|\mathrm{lc}(\check{B}_c)\| \le \|\check{B}_c\|$. We also notice that if $\#monic(\bar{\check{A}}_c) \ge d_m-1$ then $\min(\#monic(\bar{\check{A}}_c), d_m-1) = d_m-1$. If $\#monic(\bar{\check{A}}_c) < d_m-1$ then $\min(\#monic(\bar{\check{A}}_c), d_m-1) = \#monic(\check{A}_c) < d_m-1$. Without reducing the coefficient by $s(v)$, by Lemma 8.4 we have

$$\|discr(s)\bar{\check{A}}_c\| = \|(\mathrm{lc}(\check{A}_c))(discr(s)monic(\bar{\check{A}}_c))\| < (d_m-1)E|discr(s)|\|\check{A}_c\|^2.$$

The degree of v in every coefficient is now at most $2d_m-2$ after scaling because the reduction by $s(v)$ has not been performed yet. Now we reduce $discr(s)\bar{\check{A}}_c$ by $s(v)$. By Lemma 8.5 we have

$$\begin{aligned}\|discr(s)\bar{\check{A}}_c \mod s(v)\| &< (d_m-1)E|discr(s)|\|\check{A}_c\|^2(1+\|s(v)\|)^{2d_m-2-d_m+1}\\ &= (d_m-1)E|discr(s)|\|\check{A}_c\|^2(1+\|s(v)\|)^{d_m-1}.\end{aligned}$$

Similarly,

$$\|discr(s)\bar{\check{B}}_c \mod s(v)\| < (d_m-1)E|discr(s)|\|\check{A}_c\|^2(1+\|s(v)\|)^{d_m-1}.$$

Note that $\|discr(s)\bar{\check{B}}_c \mod s(v)\| = K_r(\|discr(s)\bar{\check{B}}_c \mod s(v)\|)$ because K_r is assumed to be good and invertible. Let S be the Sylvester matrix formed by the coefficients of $K_r(discr(s)\bar{\check{A}}_c)$ in x and the coefficients of $K_r(discr(s)\bar{\check{B}}_c)$ in x. S has size $\deg_{x_0}\bar{\check{A}}_c + \deg_{x_0}\bar{\check{B}}_c$ which is at most $\deg_{x_0}\check{A}_c + \deg_{x_0}\check{B}_c \le d_A+d_B$. Let us regard the algebraic integer $D_m\alpha$ as the variable v. Since v has degree at most d_m-1 in $\bar{\check{A}}_c$ and $\bar{\check{B}}_c$ and by Proposition 4.1, we have $\deg_y(K_r(\check{A}_c)) < (2d^2+1)^n$ and $\deg_y(K_r(\check{B}_c)) < (2d^2+1)^n$. $discr(s) \in \mathbb{Z}$ which does not contribute to the number of terms in $S_{i,j}$. Therefore $\#S_{i,j} \le (d_m-1+1)(2d^2+1)^n = d_m(2d^2+1)^n$. Since $\max(\|\check{A}_c\|, \|\check{B}_c\|) \le d_ch_c$, by Proposition 3.2 we have

$$\|\det S\| < (d_A+d_B)^{\frac{d_A+d_B}{2}}((d_m)(2d^2+1)^n)^{d_A+d_B}((d_m-1)E|discr(s)|d_c^2h_c^2(1+\|s(v)\|)^{d_m-1})^{d_A+d_B}.$$

Finally we compute $\det S \mod s(v)$, that is we reduce the degree of coefficients of $\det S$ in v. Let $\det S = \sum r_i y^i \in \mathbb{Z}[v][y]$. Then $\|r_i\| \leq \|\det S\|$. Since $\deg_v S_{i,j} \leq d_m - 1$ and S is a square matrix with the maximum possible size $(d_A + d_B) \times (d_A + d_B)$, $\deg_v r_i \leq (d_A + d_B)(d_m - 1)$. For each i, we apply Lemma 8.5 to r_i and $s(v)$ and get

$$\|r_i \mod s(v)\| < \|r_i\|(1 + \|s(v)\|)^{(d_A+d_B)(d_m-1)-d_m+1}.$$

Since $d_A \leq d$ and $d_B \leq d$, we have

$$\begin{aligned}
&\|\mathrm{res}_x(discr(s)K_r(\bar{\check{A}}_c), discr(s)K_r(\bar{\check{B}}_c))\| \\
\leq&(2d)^d((d_m)(2d^2+1)^n)^{2d}((d_m-1)E|discr(s)|d_c^2h_c^2(1+\|s(v)\|)^{d_m-1})^{2d}(1+\|s(v)\|)^{(2d)(d_m-1)-(d_m-1)} \\
\leq&(2d)^d((d_m)(d_m-1)(2d^2+1)^nE|discr(s)|d_c^2h_c^2)^{2d}(1+\|s(v)\|)^{(4d-1)(d_m-1)} = \mathbf{B}_{up}.
\end{aligned}$$

Therefore

$$|\mathrm{res}_x(K_r(\bar{\check{A}}_c), K_r(\bar{\check{B}}_c))|_{nmax} \leq \|\mathrm{res}_x(discr(s)K_r(\bar{\check{A}}_c), discr(s)K_r(\bar{\check{B}}_c))\| < \mathbf{B}_{up}.$$

□

Remark 8.5. The bound $\mathbf{B}_{up}$ seems to be too large. We will see more bounds like this in the coming sections. We are going to develop a prime lower bound *pmin* as we did in Chapter 4. Once *pmin* is determined, $\log_{pmin} \mathbf{B}_{up}$ is reasonably small. See Section 8.7 for examples.

Remark 8.6. For algorithm MGCD_α in Section 8.7, we don't do the algebraic integer conversion because the bounds for the number of bad and unlucky primes are not required.

8.5 Zero divisors

The reason we introduce modular arithmetic to polynomial GCD algorithms over number fields is to avoid the expression swell in $\mathbb{Q}$ caused by inverting algebraic numbers in $\mathbb{Q}(\alpha)$. We reduce the inputs modulo a series of primes then the coefficients of the inputs sit in a finite ring $\mathbb{Z}_p[z]/\langle m(z), p\rangle$. The trade off is the presence of zero divisors which creates a new problem. If there were infinitely many zero divisors for given inputs, the Euclidean algorithm over rings may still determine the correct GCD if it does not encounter zero divisors. But the good news is that the number of zero divisors is finite for a given problem. We first present the monic Euclidean algorithm over rings in Figure 8.1.

Step 2 and 4 in the Euclidean algorithm over rings compute an inverse to make a divisor monic. If the leading coefficient of the divisor is a zero divisor then Fail is returned. To overcome this we simply pick a new prime. The following example shows the possible causes to encounter zero divisors. We still use $\gcd(A, B)$ to denote the result of the Euclidean algorithm over rings with inputs A and B. In the monic Euclidean algorithm over rings,

Monic Euclidean algorithm over rings
Input: Two univariate polynomials A and B with coefficients in R where R is a commutative ring with identity 1
Output: Either Fail or the monic $\gcd(A, B)$.

1 Set $r_1 = A$, $r_2 = B$ and $i = 2$.
2 If $r_2 = 0$ and $lc(r_1)^{-1}$ exists then return $lc(r_1)^{-1}r_1$. If $r_2 = 0$ and $lc(r_1)^{-1}$ does not exist then return Fail.
3 While $r_i \neq 0$ do
  4 If $lc(r_i)^{-1}$ does not exist then return Fail. Otherwise Set $r_i = (lc(r_i)^{-1}(r_i)$.
  5 Set r_{i+1} to be the remainder of $r_{i-1} \div r_i$.
  6 Set $i = i + 1$.
7 End while loop.
8 return r_{i-1}.

Figure 8.1: Euclidean algorithm over rings

see Figure 8.1, $LC(r_i)$ and $lc(r_i)$ are interchangeable because the inputs are assumed to be univariate.

Example 8.13. *Let α be an algebraic number with the minimal polynomial $m(z) = z^2 + \frac{1}{2}$. Suppose we randomly pick the prime $p = 857$ and $K_r(B) = (525\alpha y^2 - 2205y)x^2 + 1$. Let $\beta \in \mathbb{Z}_{857}$ be an evaluation point chosen uniformly at random for y. When we run the monic Euclidean algorithm over rings on $K_r(A)(x, \beta)$ and $K_r(B)(x, \beta)$, $K_r(B)(x, \beta)$ is made to be monic first (Step 4 of the monic Euclidean algorithm), that is to compute $\frac{1}{525\alpha\beta^2 - 2205\beta}$ mod $\langle m(z), 857\rangle$. Since $\mathbb{Z}_{857}[z]/\langle m(z)\rangle$ is a finite ring, $525\alpha\beta^2 - 2205\beta$ could be a zero divisor in this ring. Recall that $525\alpha\beta^2 - 2205\beta$ is a zero divisor if $Norm(525\alpha\beta^2 - 2205\beta) \equiv 0$ mod 857. Therefore the prime and the evaluation point both play roles in making the leading coefficient to be a zero divisor. Let us consider $Norm(LC(K_r(B))$. By Corollary 8.1 we have*

$$Norm(LC(K_r(B))) = Norm(525\alpha y^2 - 2205y) = \frac{275625}{2}y^4 + 4862025y^2.$$

When $p = 3, 5$ or 7, $Norm(LC(K_r(B)))$ mod $p = 0$. If we use those primes, zero divisors are produced directly even without evaluations. On the other hand, $Norm(LC(K_r(B))) \neq 0$ mod 857. However the roots of $Norm(LC(K_r(B)))$ mod 857 are $0, 207$ and 605. Since $LC(K_r(B))(y = 0) = 0$ mod p, 0 is a bad evaluation point. Hence the leading coefficient $525\alpha y^2 - 2205y$ when evaluated at either 207 or 605 is a zero divisor. But the number of roots of $Norm(LC(K_r(B))$ still bounds the number of evaluation points causing zero divisors.

The zero divisors shown in Example 8.13 can be divided into two families. Let n be the number of division steps in the Euclidean algorithm. So $n \leq \deg_x(K_r(B))$. The first kind is the prime which divides the norm of the leading coefficients of remainders. If a prime

p is not bad and divides all rational coefficients of $Norm(lc(r_i))$ for some i, then $lc(r_i)$ is not invertible modulo p no matter what evaluation points we use. An upper bound for the number of zero divisors contributed by primes (zero divisor primes) can be derived from $\prod_{i=1}^{n} |Norm(lc(r_i))|_{nmax}$ which also bounds all bad primes defined in the previous section. But for the purpose to find a bound we can still use $\prod_{i=1}^{n} |Norm(lc(r_i))|_{nmax}$. Now let p be a prime which does not cause zero divisors for a given input. The second kind is the roots of $Norm(lc(r_i)) \in \mathbb{Z}_p[y]$. The number of zero divisors contributed by the roots (zero divisor evaluation points) and the number of bad evaluation points (defined in the next section) is bounded by $\prod_{i=1}^{n} \deg_y Norm(lc(r_i))$.

Definition 8.11. Suppose the Kronecker substitution is not bad and not unlucky and p is not a bad prime. Let $A, B \in \mathbb{Q}(\alpha)[x_0, \ldots, x_n]$ where $\deg_{x_0} A \geq \deg_{x_0} B$ and $[r_3, r_4, \ldots, r_k]$ be the polynomial remainder sequence of $r_1 = K_r(\check{A})(x, y)$ and $r_2 = K_r(\check{B})(x, y)$. Then p is not a *zero divisor prime* if p does not divide the denominator of $lc(r_i)$ and $lc(r_i)$ is invertible in $\mathbb{Z}_p(\alpha)$ for $1 \leq i \leq k$.

Theorem 8.4. *With notations used in Definition 8.11. For* $1 \leq i \leq k$, *if* $Norm(lc(r_i)) \neq 0 \mod p$, *then* $lc(r_i)$ *is invertible in* $\mathbb{Z}_p(\alpha)$.

Proof. By Corollary 8.1, $Norm(lc(r_i)) = \text{res}_z(m(z), lc(r_i))$ where $m(z)$ is the minimal polynomial for α. We extend $m(z)$ from $\mathbb{Q}[z]$ to $\mathbb{Q}[y][z]$. Note that $lc(r_i) \in \mathbb{Q}[y][z]$. Since $m(z)$ is the minimal polynomial, $\gcd(m(z), lc(r_i)) = 1$. By the generalization of Bezout's identity to polynomials over an arbitrary commutative ring (with identity in our case), there exist polynomials $S, T \in \mathbb{Q}[y][z]$ such that

$$S \cdot lc(r_i) + T \cdot m(z) = \text{res}_z(m(z), lc(r_i)).$$

Since $Norm(lc(r_i)) \neq 0 \mod p$, $\text{res}_z(m, lc(r_i)) \neq 0 \mod p$. We have the equation

$$\frac{S}{\text{res}_z(m(z), lc(r_i))} \cdot lc(r_i) + \frac{T}{\text{res}_z(m(z), lc(r_i))} \cdot m(z) = 1 \mod p.$$

Hence $\frac{S}{Norm(lc(r_i))}$ is the inverse of $lc(r_i)$ modulo p. □

Definition 8.12. Suppose p is not a bad prime and not a zero divisor prime. With same notations used in Definition 8.11 let $\beta \in \mathbb{Z}_p$ be an evaluation point for y. β is not a *zero divisor evaluation point* if $lc(r_i)(\beta) \in \mathbb{Q}[z]$ is invertible modulo p for $1 \leq i \leq n$.

If we apply the monic Euclidean algorithm over rings on $K_r(\check{A})(x, y), K_r(\check{B})(x, y) \in R[x]$ (R is the coefficients ring) to derive bounds for the number of zero divisors primes and the number of zero divisor evaluation points then the leading coefficient in every remainder is a fraction in $\mathbb{Z}[y, z]/\mathbb{Z}[y, z]$ which is difficult to handle. This caused us much grief so

Monic subresultant PRS
Input: Two univariate polynomials A and B in x with coefficients in R where R is a commutative ring with identity 1 and $\deg_x A \geq \deg_x B$.
Output: Either Fail or the monic $\gcd(A, B)$.

1 Set $r_1 = A$, $r_2 = B$, $i = 2$ and $c_1 = lc(r_1)$.
2 If c_1 is not invertible, then return Fail. If $r_2 = 0$ then return $c_1^{-1} r_1$.
3 While $r_i \neq 0$ do
  4 Let $d_i = \deg_x r_{i-1} - \deg_x r_i$ and $c_i = lc(r_i)$.
  5 If inverting c_i fails, then return Fail.
  6 If $i = 2$ then $h_i = -1$ else $h_i = -c_{i-1}^{d_{i-1}} (h_{i-1}^{d_{i-1}-1})^{-1}$.
  7 If $i = 2$ then $b_i = (-1)^{d_i+1}$ else $b_i = -c_{i-1} h_i^{d_i}$.
  8 Let $\bar{r} = prem(r_{i-1}, r_i)$ where $prem$ denotes the pseudo division.
  Note: No inverse computation is performed in $prem$. See Figure 1.1.
  9 Let $r_{i+1} = \bar{r} b_i^{-1}$ and $i = i + 1$.
10 Return $r_{i-1} = c_{i-1}^{-1} r_{i-1}$.
Note: r_{i-1} is the last non-zero remainder and c_{i-1} is invertible by step 5.

Figure 8.2: Monic subresultant PRS algorithm

we decided to use the monic subresultant PRS algorithm instead to compute the GCD in $\mathbb{Z}(\alpha)[x]$. See Figure 8.2.

For the monic subresultant PRS algorithm on inputs $K_r(\check{A})(x, y)$ and $K_r(\check{B})(x, y)$ we treat inputs as univariate polynomials in x. In step 2 we invert the leading coefficient of $K_r(\check{A})(x, y)$. Inverting is also required in step 5. In step 6 and step 9 we compute $(h_{i-1}^{d_{i-1}-1})^{-1}$ and b_i^{-1} which involve inverse computations. But it is obvious that the denominator and the numerator of h_i is the product of leading coefficients of remainders in previous iterations if we don't do cancellation. And b_i depends on h_i and c_i. For example, $h_2 = -1$, $h_3 = \frac{-lc(r_2)^{d_2}}{(-1)^{d_2-1}}$, $h_4 = \frac{lc(r_3)^{d_3}(-1)^{(d_2-1)(d_3-1)}}{lc(r_2)^{(d_2)(d_3-1)}}$,...,etc. Therefore if step 5 does not fail, step 6 and step 9 cannot fail and the pseudo remainder in step 8 does not have zero divisor problem. If $p | Norm(lc(r_i))$ for some $i < k$, provided that p is not bad, then p is a zero divisor prime. If the numerator or the denominator of h_i evaluated at β is zero modulo p, then $Norm(lc(r_i))(\beta) \mod p = 0$ for some $i < k$. Hence if we avoid zero divisor primes and for each prime, which is not a zero divisor prime, we avoid to use zero divisor evaluation points for the leading coefficient of each remainder in the subresultant PRS, then the monic subresultant PRS algorithm outputs the monic remainder r_k.

In the remaining part of this section, we derive upper bounds for the number of zero divisor evaluation points and the number of zero divisor primes by the definition of subresultant polynomial. First we review the subresultant PRS defined by the subresultant polynomial.

Let $K_r(\check{A}) = \sum_{i=0}^{d_A} a_i x^i$ and $K_r(\check{B}) = \sum_{i=0}^{d_B} b_i x^i$ where $d_A \geq d_B \geq 1$ and $a_i, b_i \in \mathbb{Z}[z][y]$. The Sylvester matrix of $K_r(\check{A})$ and $K_r(\check{B})$ is

$$S = \begin{bmatrix} a_{d_A} & 0 & & 0 & b_{d_B} & 0 & & 0 \\ a_{d_A-1} & a_{d_A} & & 0 & b_{d_B-1} & b_{d_B} & & 0 \\ \vdots & a_{d_A-1} & \ddots & 0 & \vdots & b_{d_B-1} & \ddots & 0 \\ a_1 & \vdots & & a_{d_A} & b_1 & \vdots & & b_{d_B} \\ a_0 & a_1 & & a_{d_A-1} & b_0 & b_1 & & b_{d_B-1} \\ 0 & a_0 & & \vdots & 0 & b_0 & & \vdots \\ 0 & 0 & \ddots & a_1 & 0 & 0 & \ddots & b_1 \\ 0 & 0 & & a_0 & 0 & 0 & & b_0 \end{bmatrix}. \tag{8.1}$$

By the definition of polynomial resultant $\mathrm{res}_x(K_r(\check{A}), K_r(\check{B})) = \det S$. Let $S(i,j)$ be the $(d_A + d_B - 2j) \times (d_A + d_B - 2j)$ submatrix of S obtained by deleting

1. columns $d_B - j + 1$ to d_B;
2. columns $d_A + d_B - j + 1$ to $d_A + d_B$;
3. row $d_A + d_B - 2j$ to $d_A + d_B$ except the row $d_A + d_B - i - j$.

Definition 8.13. The j-th subresultant of $K_r(\check{A})$ and $K_r(\check{B})$ is the polynomial

$$S(j, K_r(\check{A}), K_r(\check{B})) = \det S(0,j) + \det S(1,j)x + \cdots + \det S(j,j)x^j \in \mathbb{Z}[y,z][x].$$

Note that $\det S(i,j)$ is not reduced by $\check{m}(z)$ and $\det S(i,j) \in \mathbb{Z}[y,z]$. If $\det S(j,j) \equiv 0 \mod m(z)$, $\deg_x S(j, K_r(\check{A}), K_r(\check{B})) < j$. But it does not affect the calculation of the upper bounds. Hence we assume $\det S(j,j) \not\equiv 0 \mod m(z)$. It is clear that the size of $S(j,j)$ increases as j decreases and $\det S(0,0) = \det S$. We claim that the upper bound of the number of zero divisor evaluation points and the upper bound of the number of zero divisor primes for $Norm(\det S)$ also bound the number of zero divisor evaluation points and the number of zero divisor primes for $Norm(\det S(i,j))$ respectively for $0 \leq i \leq j$ and $0 \leq j \leq d_B$. Recall that

$$Norm(\det S) = \mathrm{res}_z(\check{m}(z), \mathrm{res}_x(K_r(\check{A}), K_r(\check{B})) \mod \check{m}(z)).$$

Theorem 8.5. *Let $K_r(\check{A}), K_r(\check{B}) \in \mathbb{Z}_p[z]/\langle m(z)\rangle[y][x]$ and p be a prime which is not bad and not a zero divisor prime. Then there are at most $2d(d+1)(2d^2+1)^n d_m + dd_m$ zero divisor evaluation points.*

Proof. We first determine an upper bound for the number of zero divisor evaluation points for $\det S(0,0) = \det S$. Note that $\det S(0,0)$ is the leading coefficient of $S(0, K_r(\check{A}), K_r(\check{B}))$.

Suppose p is not bad and not a zero divisor prime. The first step is to compute $\deg_y \det S = \deg_y \operatorname{res}_x(K_r(\check{A}), K_r(\check{B}))$ where S is a $(d_A + d_B) \times (d_A + d_B)$ Sylvester matrix. In the first d_B rows, $\deg_y S_{i,j} \leq \deg_y K_r(\check{A})$. In the last d_A rows, $\deg_y S_{i,j} \leq \deg_y K_r(\check{B})$. Hence $\deg_y \det S \leq d_B \deg_y K_r(\check{A}) + d_A \deg_y K_r(\check{B})$.

The second step is to compute $\operatorname{res}_z(\check{m}(z), \operatorname{res}_x(K_r(\check{A}), K_r(\check{B})) \mod \check{m}(z))$. The first $\deg_z \operatorname{res}_x(K_r(\check{A}), K_r(\check{B})) \mod \check{m}(z)$ rows contain only rationals from the coefficients of $\check{m}(z)$. In the last $\deg_z \check{m}(z)$ rows, the degree of y in every entry is bounded by

$$\deg_y \operatorname{res}_x(K_r(\check{A}), K_r(\check{B})) \mod \check{m}(z) \leq d_B \deg_y K_r(\check{A}) + d_A \deg_y K_r(\check{B}).$$

Therefore,

$$\deg_y Norm(\det S) \leq \deg_z \check{m}(z)(d_B \deg_y K_r(\check{A}) + d_A \deg_y K_r(\check{B})).$$

Since $d_A \leq d$, $d_B \leq d$, $\deg_y K_r(\check{A}) \leq (2d^2+1)^n$ and $\deg_y K_r(\check{B}) \leq (2d^2+1)^n$, we have

$$\deg_y Norm(\det S) \leq 2d(2d^2+1)^n d_m.$$

In other words, there are at most $2d(2d^2+1)^n d_m$ zero divisor evaluation points for the norm of the leading coefficient of $S(0, K_r(\check{A}), K_r(\check{B}))$. For the leading coefficients $\det S(j,j)$ where $0 < j \leq d_B$, the size of the matrix $S(j,j)$ is $(d_A + d_B - 2j) \times (d_A + d_B - 2j)$ which is less than the size of S and the degree in each entry of $S(j,j)$ is bounded by the same quantity and therefore $\deg_y Norm(\det S(j,j)) \leq \deg_y Norm(\det S)$ for $0 < j \leq d_B$. Since there are at most $d_B + 1$ division steps, the total number of zero divisor evaluation points contributed by the while loop in the monic subresultant PRS algorithm is bounded by $(2d(2d^2+1)^n d_m)(d_B+1) \leq 2d(d+1)(2d^2+1)^n d_m$.

Finally we bound the number of roots for y in $Norm(lc(r_1))$. Since $Norm(lc(r_1)) = \operatorname{res}_z(\check{m}(z), lc(r_1))$, the number of roots is bounded by the degree of the determinant of the Sylvester matrix of $\check{m}(z)$ and $lc(r_1)$ in y. Since $\deg_z lc(r_1) < d_m$, the size of this Sylvester matrix is less than $d_m + d_m$. But the entries in the first $\deg_z lc(r_1)$ columns are integers. The degree of each entry in y in this Sylvester matrix is bounded by $\deg_y lc(r_1) \leq d$. Therefore $\deg_y \operatorname{res}_z(\check{m}(z), lc(r_1)) \leq d_m d$ and the total number of zero divisor evaluation points is bounded by

$$(2d(2d^2+1)^n d_m)(d_B+1) + dd_m \leq 2d(d+1)(2d^2+1)^n d_m + dd_m.$$

□

Recall that $h > \max(\|\check{A}\|, \|\check{B}\|)$. We have the following theorem.

Theorem 8.6. *Let* $\mathbf{B}_{zdp} = ((2d_m)^{d_m}(d_m(d+1))^{2d_m}h^{2d_m})((2d_m)^{d_m}(2d(2d^2+1)^n)^{2d_m}W^{2d_m})^{d+1}$ *where*

$$W = (2d)^d((2d^2+1)^n d_m)^{2d}h^{2d}(|D_m| + \|\check{m}(z)\|)^{\gamma+1} \quad \textit{and} \quad \gamma = (d_A + d_B)(d_m - 1) + 1.$$

There are at most $\log_{pmin} \mathbf{B}_{zdp}$ *zero divisor primes* p *with* $p \geq pmin$.

Proof. Let $D_m = lc(\check{m}(z))$. Recall that $h > \max(\|\check{A}\|, \|\check{B}\|)$. By the choice of K_r, see Remark 4.1, we have $h > \|K_r(\check{A})\|$ and $h > \|K_r(\check{B})\|$. We first compute $\|Norm(lc(r_1))\|$, that is to compute the height of $\mathrm{res}_z(\check{m}(z), lc(r_1))$. Let S' be the Sylvester matrix formed by the coefficients of $\check{m}(z)$ and $lc(r_1)$. Then $\det S' = \mathrm{res}_z(\check{m}(z), lc(r_1))$. Since $\deg_z \check{m}(z) = d_m$ and $\deg_z lc(r_1) < d_m$, the dimension of S' is bounded by $2d_m$. We also have $\#\det(S') \leq (d_m - 1+1)(\deg_y lc(r_1)+1) \leq d_m(d+1)$. By Proposition 3.2, we have $\|\det S'\| \leq (2d_m)^{d_m}(d_m(d+1))^{2d_m}h^{2d_m}$. Therefore

$$\|Norm(lc(r_1))\| \leq (2d_m)^{d_m}(d_m(d+1))^{2d_m}h^{2d_m}.$$

Recall that the Sylvester matrix S contains elements in $\mathbb{Z}[z][y]$. Since $\deg_y K_r(\check{A}) \leq (2d^2+1)^n$, $\deg_y K_r(\check{B}) \leq (2d^2+1)^n$, $\deg_z K_r(\check{A}) < d_m$ and $\deg_z K_r(\check{B}) < d_m$, $\#S_{i,j} < (2d^2+1)^n(d_m - 1 + 1)$. Therefore by 3.2, we have

$$\|\det S\| \leq (d_A + d_B)^{\frac{d_A+d_B}{2}}((2d^2+1)^n d_m)^{d_A+d_B}h^{d_A+d_B}.$$

Since $\deg_z S_{ij} \leq \deg_z m(z) - 1$, $\deg_z \det S \leq (d_A + d_B)(d_m - 1)$. Let $\gamma = (d_A + d_B)(d_m - 1) + 1$. By the pseudo division and Lemma 8.5, we have

$$\begin{aligned}\|D_m^\gamma \det S \mod \check{m}(z)\| &\leq \|D_m^\gamma \det S\|(1 + \|\check{m}(z)\|/|D_m|)^\gamma \\ &= |D_m^\gamma|\|\det S\|(1 + \|\check{m}(z)\|/|D_m|)^\gamma \\ &= \|\det S\|(|D_m| + \|\check{m}(z)\|)^\gamma.\end{aligned}$$

Hence by Proposition 3.2 we have

$$\begin{aligned}&\|D_m^\gamma \det S \mod \check{m}(z)\| \\ &\leq (d_A + d_B)^{\frac{d_A+d_B}{2}}((2d^2+1)^n d_m)^{d_A+d_B}h^{d_A+d_B}(|D_m| + \|\check{m}(z)\|)^\gamma.\end{aligned}$$

The final step is to derive an upper bound for the numerator magnitude of

$$Norm(D_m^\gamma \det S \mod \check{m}(z)) = \mathrm{res}_z(\check{m}(z), D_m^\gamma \det S \mod \check{m}(z)).$$

Let V be the Sylvester matrix of $\check{m}(z)$ and $D_m^\gamma \det S \mod \check{m}(z)$. Let $d_S = \deg_z(D_m^\gamma \det S \mod \check{m}(z))$. By the property of determinant, V is a $(d_S + d_m) \times (d_S + d_m)$ matrix in $\mathbb{Z}[y]$

and

$$\begin{aligned}\|V_{ij}\| &\le \max\{\|D_m^\gamma \det S \mod \check{m}(z)\|, \|\check{m}(z)\|\} \\ &\le (d_A+d_B)^{\frac{d_A+d_B}{2}}((2d^2+1)^n d_m)^{d_A+d_B} h^{d_A+d_B}(|D_m|+\|\check{m}(z)\|)^\gamma.\end{aligned}$$

Since $\deg_y \det S \le d_B \deg_y K_r(\check{A}) + d_A \deg_y K_r(\check{B}) \le 2d(2d^2+1)^n$ we have

$$\deg_y D_m^\gamma \det S \mod \check{m}(z) \le 2d(2d^2+1)^n$$

which is the term bound. Therefore by Proposition 3.2 again with $d_A, d_B \le d$ and $d_S \le d_m - 1$ we have

$$\|\det V'\| < (2d_m)^{d_m}(2d(2d^2+1)^n)^{2d_m} W^{2d_m},$$

where $W = (2d)^d((2d^2+1)^n d_m)^{2d} h^{2d}(|D_m| + \|\check{m}(z)\|)^\gamma$. Let $\mathbf{B}_{prs}$ be the quantity on the right hand side of the above inequality. We also notice that $S(i,j)$ is a submatrix of S by deleting rows and columns. The height of each entry in $S(i,j)$ is bounded by h and the size of $S(i,j)$ is bounded by $d_A + d_B$. Hence the numerator magnitude of $Norm(\det S(i,j) \mod \check{m}(z))$ is also bounded by $\mathbf{B}_{prs}$. In the monic subresultant algorithm on inputs $K_r(\check{A})$ and $K_r(\check{B})$, there are at most $d_B + 1 \le d+1$ division steps. Let $pmin$ be a positive integer. If we consider primes $p > pmin$ then there are at most $\log_{pmin} \mathbf{B}_{zdp}$ zero divisor primes with input $K_r(\check{A})$ and $K_r(\check{B})$, where

$$\begin{aligned}\mathbf{B}_{zdp} =& ((2d_m)^{d_m}(d_m(d+1))^{2d_m} h^{2d_m})\mathbf{B}_{prs}^{d+1} \\ =& ((2d_m)^{d_m}(d_m(d+1))^{2d_m} h^{2d_m})((2d_m)^{d_m}(2d(2d^2+1)^n)^{2d_m} W^{2d_m})^{d+1}.\end{aligned}$$

and $W = (2d)^d((2d^2+1)^n d_m)^{2d} h^{2d}(|D_m| + \|\check{m}(z)\|)^\gamma$. □

Remark 8.7. We mention again that the bound $\mathbf{B}_{zdp}$ seems large but $\log_{pmin} \mathbf{B}_{zdp}$ is small once $pmin$ is determined. See Section 8.7 for more details.

8.6 Bad and unlucky evaluation points

In this section we derive the upper bounds for the number of bad evaluation points and the number of unlucky evaluation points.

Definition 8.14. Suppose the prime p is not bad and not unlucky. Let $\beta \in \mathbb{Z}_p$ be an evaluation point chosen uniformly at random. If $LC(K_r(\check{A}))(x,\beta) = 0$ or $LC(K_r(\check{B}))(x,\beta) = 0$ then β is a *bad evaluation point*. If the Euclidean algorithm successfully determines $\bar{G} = \gcd(K_r(\bar{\check{A}})(x,\beta), K_r(\bar{\check{B}})(x,\beta))$ without failure and $\deg_x \bar{G} > 0$, then β is an *unlucky evaluation point*. If β is not bad and not unlucky, we call it *good*.

Theorem 8.7. *Let p be a radical prime which is not bad and not unlucky. If β is chosen uniformly at random from $\mathbb{Z}_p$, then*

(i) $\text{Prob}[\beta \text{ is bad}] < \frac{2(2d^2+1)^n}{p}$ *and*

(ii) $\text{Prob}[\beta \text{ is bad or unlucky}] < \frac{2(2d^2+1)^n + 2d(2d^2+1)^n}{p}$.

Proof. By Lemma 8.1 we have

$$\begin{aligned}\text{Prob}[\beta \text{ is bad}] &= Prob(LC(K_r(\check{A}))(x,\beta)LC(K_r(\check{B}))(x,\beta) = 0) \\ &\leq \frac{\deg_y(LC(K_r(\check{A})) + \deg_y(LC(K_r(\check{B}))}{p} \\ &\leq \frac{\deg_y K_r(\check{A}) + \deg_y K_r(\check{B})}{p} < \frac{2(2d^2+1)^n}{p}.\end{aligned}$$

(i) is proved. For (ii), from the proof of Lemma 2.7 we have

$$\text{Prob}[\beta \text{ is bad or unlucky}] \leq \frac{\deg_y K_r(LC(\check{A}))K_r(LC(\check{B}))}{p} + \frac{\#\ unlucky\ evaluation\ points}{p}.$$

Therefore we need to determine an upper bound for the number of unlucky evaluation points. Since $\check{m}(z)$ is the semi-associate of the minimal polynomial it is irreducible and therefore squarefree over $\mathbb{Q}$. By Lemma 8.8 $\langle \check{m}(z) \rangle$ is a radical ideal. The prime p we pick is assumed to be a radical prime, and therefore $\langle \check{m}(z) \mod p \rangle$ is still a radical ideal. By Lemma 8.8 again we have $\check{m}(z) \mod p = m_1(z) \cdots m_k(z) \mod p$ where $m_i(z) \neq m(z)_j$ if $i \neq j$ and $m_i(z)$ is irreducible over $\mathbb{Z}_p$ for $1 \leq i \leq k$. By the Chinese remainder theorem we have the decomposition

$$\mathbb{Z}_p[z]/\langle \check{m}(z) \rangle \cong \mathbb{Z}_p[z]/\langle m_1(z) \rangle \times \cdots \times \mathbb{Z}_p[z]/\langle m_k(z) \rangle.$$

We extend this isomorphism to a bivariate polynomial ring and get

$$\mathbb{Z}_p[z]/\langle \check{m}(z) \rangle[x,y] \cong \mathbb{Z}_p[z]/\langle m_1(z) \rangle[x,y] \times \cdots \times \mathbb{Z}_p[z]/\langle m_k(z) \rangle[x,y].$$

We note that each component on the right hand side of above equation is a polynomial ring over the field $\mathbb{Z}_p[z]/\langle m_i(z) \rangle$ because m_i is irreducible modulo p for $1 \leq i \leq k$. Suppose β is an unlucky evaluation point and the monic subresultant PRS algorithm successfully outputs $\bar{g} = \gcd(K_r(\bar{\check{A}})(x,\beta), K_r(\bar{\check{B}})(x,\beta)) \in \mathbb{Z}_p[z]/\langle \check{m}(z) \rangle[x]$. Since β is unlucky and $\bar{g}$ is monic, $\deg_x \bar{g} = \deg_x \bar{g} \mod m_i(z) \geq 1$ for $1 \leq i \leq k$. By Lemma 2.4 $\deg_x \bar{g} \mod m_i(z) \geq 1$ is equivalent to $R_i(\beta) = \text{res}_x(K_r(\bar{\check{A}})(x,\beta), K_r(\bar{\check{B}})(x,\beta)) \equiv 0 \mod m_i(z)$. Hence β is a root of $R_i(y)$. This is true for $1 \leq i \leq k$. By our choice of Kronecker substitution, $\deg_y R_i(y) < 2d(2d^2+1)^n$. Note that the bound for $\deg_y R_i(y)$ can be obtained by the same way as we did in Proposition 4.1. Hence there are at most $2d(2d^2+1)^n$ roots for each $R_i(y)$. Now

we apply the Chinese remaindering theorem to $\{R_i : 1 \leq i \leq k\}$ with moduli $\{m_i\}_{1\leq i\leq k}$ and obtain a polynomial $r \in \mathbb{Z}_p[z]/\langle \check{m}(z)\rangle[y]$ and $\deg_y r(y) \leq 2d(2d^2+1)^n$. Therefore $\deg_x \gcd(\bar{\check{A}}(x,\beta), \bar{\check{B}}(x,\beta)) \geq 1$ implies $r(\beta) = 0$ and the number of unlucky evaluation points is bounded by the number of roots of $r(y)$. But we only choose evaluation points from $\mathbb{Z}_p$ and thus $r(y)$ has at most $2d(2d^2+1)^n$ roots in $\mathbb{Z}_p$. So we have

$$\text{Prob}[\beta \textit{ is bad or unlucky}] < \frac{2(2d^2+1)^n + 2d(2d^2+1)^n}{p}.$$

□

Remark 8.8. In the proof of above theorem, the polynomial $r(y)$ has coefficients in a finite ring with zero divisors and therefore it may have more than $2d(2d^2+1)^n$ roots. But the number of roots of $r(y)$ in $\mathbb{Z}_p$ is still bounded by $2d(2d^2+1)^n$.

Example 8.14. *Let $m(z) = z^2+13z+6$ be the minimal polynomial of the algebraic number α. Consider the polynomial $r(y) = y^3 + \alpha y^2 + 3y^2 + 3\alpha y + 11y + 11\alpha \in \mathbb{Q}(\alpha)[y]$. If we pick the prime $p = 17$, then $m(z) \equiv (z+8)(z+5) \mod p$. $r(y)$ has two roots in $\mathbb{Z}_p[z]/\langle z+5\rangle$ which are $5, 9$ and three roots in $\mathbb{Z}_p[z]/\langle z+8\rangle$ which are $5, 8, 9$. The number of roots cannot be greater than 3 since $\mathbb{Z}_p[z]/\langle z+5\rangle$ and $\mathbb{Z}_p[z]/\langle z+8\rangle$ are both fields and $\deg_y r(y) = 3$. But $r(y)$ has six roots in $\mathbb{Z}_p[z]/\langle m(z)\rangle$ which are $5, 9, 16\alpha, 10\alpha+4, 7\alpha+10, 6\alpha+5$. By the extended Euclidean algorithm we have $(z+8)S + (z+5)T = 1 \mod p$ for some $S, T \in \mathbb{Z}_p[z]$. By the Chinese remainder theorem, all those six roots can be obtained from $E_1(z+8)S + E_2(z+5)T \mod p$ where $E_1 \in \{5, 9\}$ and $E_2 \in \{5, 8, 9\}$. However if we restrict the roots to be in $\mathbb{Z}_p$ then there are only two roots $\{5, 9\}$.*

8.7 Simplified algorithm

We present the simplified algorithm MGCD_α in this section. The algorithm contains two parts. MGCD_α is the main routine which calls PGCD_α several times to compute enough modular GCD images then it performs the Chinese remaindering and rational number reconstruction to get the GCD. PGCD_α uses Ben-Or/Tiwari interpolation to compute one modular image modulo a prime p.

The probability that our algorithm does not encounter bad, unlucky or zero divisor evaluation points can be estimated as follows. Let U be the bound on the number of bad, unlucky and zero divisor evaluation points and τ be an upper bound for the number of univariate GCD images. We need 2τ consecutive points which are not bad, not unlucky and not zero divisor evaluation points. Let ω be a generator of $\mathbb{Z}_p^*$. Suppose ω^k is a bad or unlucky or zero divisor evaluation point where $s \leq k < s + 2\tau$ where $1 \leq s \leq p-1$. The union of all segments of length 2τ contains the point ω^k has length $4\tau - 1$. We can not use any point in this union to determine the correct feedback polynomial. The worst

case is that for all bad, unlucky and zero divisor evaluation points, their corresponding segments of length $4\tau - 1$ do not overlap. Hence there are at most $U(4\tau - 1)$ points in $p - 1$ points we cannot use. The number of bad and unlucky evaluation points is bounded by $2(2d^2+1)^n + 2d(2d^2+1)^n$ and the number of zero divisor evaluation points is bounded by $2d(d+1)(2d^2+1)^n d_m + dd_m$. Let

$$B_e = 2d(d+1)(2d^2+1)^n d_m + dd_m + 2(2d^2+1)^n + 2d(2d^2+1)^n.$$

Then we have $U \leq B_e$.

Theorem 8.8. *Suppose p is a radical, not bad, not unlucky and not zero divisor prime. Let $X \geq \frac{4\tau B_e}{p}$. Then*

$$\text{Prob}[2\tau \textit{ consecutive points fail to determine the feedback polynomial}] < \frac{4\tau B_e}{p} \leq \frac{1}{X},$$

for some positive number X. So if we choose $p \geq 4X\tau B_e$, then probability that $PGCD_\alpha$ fails is at most $\frac{1}{X}$.

Now we construct a set $S = \{p_1, p_2, \ldots, p_N\}$ which consists of smooth primes p_i such that $p_i > 4\tau B_e$ for $1 \leq i \leq N$. The number of primes which are not radical was derived in Lemma 8.10. We set $\mathbf{B}_{nr} = (1 + d_m^{d_m})|m|_{nmax}^{d_m}$ which bounds the number of primes which are not radical. This quantity only depends on the degree and the coefficients of the minimal polynomial $m(z)$ and is generally negligible, but we still include it here. The number of primes in S can be obtained by computing

$$N = Y \lceil \log_{4X\tau B_e} \mathbf{B} \rceil > Y \log_{pmin} \mathbf{B}$$

where $pmin = \min S$, $\mathbf{B} = \mathbf{B}_{bp} \cdot \mathbf{B}_{up} \cdot \mathbf{B}_{zdp} \cdot \mathbf{B}_{nr}$ and Y is some positive number.

Theorem 8.9. *Let S be the set of primes constructed above and Y be a positive number. Let p be chosen uniformly at random from S. Then*

$$\text{Prob}[p \textit{ is bad or unlucky or a zero divisor prime or a non-radical prime}] \leq \frac{\log_{pmin} B}{N} < \frac{1}{Y}.$$

Theorem 8.10. *For given $X > 0$ and $Y > 0$, we construct a set S of smooth primes as described above. Let p be chosen uniformly at random from S, s be chosen uniformly at random from $0 < s \leq p-1$ and ω be a random generator of $\mathbb{Z}_p^*$. Let $E = \{\omega^{s+j} : 0 \leq j < 2\tau\}$ be 2τ consecutive evaluation points. Then the probability that p is a radical, not bad, not unlucky and not zero divisor prime and all points in E are not bad, not unlucky and not zero divisor evaluation points is greater than $(1 - \frac{1}{X})(1 - \frac{1}{Y})$. In particularly if $X = Y = 4$ then the probability is at least $1/2$.*

This simplified algorithm may requires primes larger than 127 bits even for small inputs. In this case it requires the multi-precision arithmetic hence is less effective. But it is still worthwhile to get a sense for how large S might be because those bounds seem to be extremely large. We first review all bounds.
Bad primes:

$$\mathbf{B}_{bp} = d_c^2 h_c^2$$

where $h_c = \max(|A_c|_{nmax}, |B_c|_{nmax})$ and $d_c = \max(den(A_c), den(B_c))$.
Unlucky primes:

$$\mathbf{B}_{up} = (2d)^d((d_m)(d_m-1)(2d^2+1)^n E|discr(s)|d_c^2 h_c^2)^{2d}(1+\|s(v)\|)^{(4d-1)(d_m-1)}$$

where $E = e^{(n+1)d} d_m (d_m-1)^{\frac{d_m-1}{2}} \|s(v)\|_2^{d_m-1} |discr(s)|^{-\frac{1}{2}} \sum_{i=0}^{d_m-1} \|s(v)\|_2^i$ and $\|\cdot\|_2$ denotes the 2-norm.
Zero divisor primes:

$$\mathbf{B}_{zdp} = ((2d_m)^{d_m}(d_m(d+1))^{2d_m} h^{2d_m})((2d_m)^{d_m}(2d(2d^2+1)^n)^{2d_m} W^{2d_m})^{d+1}$$

where

$$W = (2d)^d((2d^2+1)^n d_m)^{2d} h^{2d}(|D_m| + \|\check{m}(z)\|)^{\gamma+1} \quad \text{and} \quad \gamma = (d_A + d_B)(d_m - 1) + 1.$$

Non-radical primes:

$$\mathbf{B}_{nr} = |m|_{nmax}^{d_m} + (d_m |m|_{nmax})^{d_m} = (1 + d_m^{d_m}) |m|_{nmax}^{d_m}.$$

The following example shows $\log_{pmin} \mathbf{B}$ is reasonably small.

Example 8.15. *Let* $m(z) = z^4 - \frac{7}{3}z^3 + \frac{11}{2}z^2 - \frac{14}{5}z + 1$ *be the minimal polynomial of the algebraic number* α. *Hence* $D_m = 30$ *and* $d_m = 4$. *The semi-associate of* $m(z)$ *is* $\check{m}(z) = 30z^4 - 70z^3 + 165z^2 - 84z + 30$ *and* $\|\check{m}(z)\| = 165$. *Let* $\check{A}, \check{B} \in \mathbb{Z}(\alpha)[x_0, \ldots, x_4]$, *then* $n = 4$. *We also assume that* $\|\check{A}\| = \|\check{B}\| = 10000$, $\tau = 100$ *and* $\deg_{x_i} A = \deg_{x_i} B = 5$ *for* $0 \le i \le 4$, *hence* $d = 5$. *Let* A_c *and* B_c *be the converted* $\check{A}$ *and* $\check{B}$ *respectively.* $s(v) = v^4 - 70v^3 + 4950v^2 - 75600v + 810000$ *is the minimal polynomial of the algebraic integer* $D_m\alpha$. *Let* $h_c = \max(|A_c|_{nmax}, |B_c|_{nmax}) \le \max(\|\check{A}\|, \|\check{B}\|) = 10000$ *and* $d_c = \max(den(A_c), den(B_c)) \le D_m^{d_m - 1} = 27000$. *We pick* $X = 4$ *and* $Y = 4$ *so that* $PGCD_\alpha$ *computes the correct modular image with probability at least* $1/2$. *Now we have all parameters we need. We first compute* $B_e = 1704830672$. *So the lower bound for primes is* $4 \cdot 4 \cdot \tau \cdot Be = 2727729075200$ *which is a* 42 *bits integer.*

$$\mathbf{B}_{zdp} \approx 1.89 \times 10^{9579}, \quad \mathbf{B}_{up} \approx 4.46 \times 10^{1170}, \mathbf{B}_{bp} = 7.29 \times 10^{16}, \mathbf{B}_{nr} = 190488560625.$$

We pick $\mathbf{B} = \mathbf{B}_{bp} \cdot \mathbf{B}_{up} \cdot \mathbf{B}_{zdp} \cdot \mathbf{B}_{nr} = 1.17 \times 10^{10778}$. *Therefore* $\log_{4\cdot 4\cdot \tau \cdot Be} \mathbf{B} \approx 866.70$ *and* $|S| = Y \cdot \lceil 866.70 \rceil = 4 \cdot \lceil 867 \rceil = 3468$.

We square the degree of each variable in inputs to get $d = 25$*, then the lower bound for primes is* $4 \cdot 4 \cdot \tau \cdot Be = 20581353822024563200$ *which is a* 65 *bits integer.* $|S| = 55708$ *and* $55708/3468 \approx 16$*. Let* $d = 5$ *again. If we double the length of the height then we get* $\|\check{A}\| = \|\check{B}\| = 10^8$*, then the lower bound for primes is* $4 \cdot 4 \cdot \tau \cdot Be = 2727729075200$ *which is a* 42 *bits integer and* $|S| = 4124$.

We assume that monomials are in pure lexicographical order $x_0 > x_1 > \cdots > x_n$, x_0 is the main variable and $F \in \mathbb{Q}(\alpha)[x_0, \ldots, x_n]$. Let $LC(F)$ denote the leading coefficient of F with respect to x_0 hence $LC(F) \in \mathbb{Q}(\alpha)[x_1, \ldots, x_n]$. Let $lc(F)$ denote the leading coefficient of F with respect to $x_0, x_1, \ldots, x_n$ hence $lc(F) \in \mathbb{Q}(\alpha)$. Suppose $A, B \in \mathbb{Q}(\alpha)[x_0, \ldots, x_n]$ and $\Gamma = \gcd(LC(A), LC(B)) \in \mathbb{Q}(\alpha)[x_1, \ldots, x_n]$, $H = \sum h_i H_i = \Delta G$ where H_i is a monomial in x_0, α and $\Delta = \Gamma / LC(G)$. Γ has one less variable than A and B and we recursively use MGCD_α to compute Γ.

Algorithm $\mathrm{MGCD}_\alpha(\ A,\ B,\ \tau,\ m(z)\ )$

Inputs $A, B \in \mathbb{Q}(\alpha)[x_0, x_1, \ldots, x_n]$ where $n > 0$, α is an algebraic number with monic minimal polynomial $m(z) \in \mathbb{Q}[z]$, A and B are primitive in x_0, $\deg_{x_0} A > 0$, $\deg_{x_0} B > 0$. The monic minimal polynomial $m(z) \in \mathbb{Q}[z]$ for α. A term bound τ satisfying $\tau \geq \max \#h_i$.

Output $G = \gcd(A, B)$.

1 Let $\check{A}$, $\check{B}$ and $\check{m}(z)$ be the semi-associates of A, B and $m(z)$ respectively.

2 Compute $\Gamma = \gcd(LC(\check{A}), LC(\check{B}))$ in $\mathbb{Q}(\alpha)[x_1, \ldots, x_n]$ by MGCD_α recursively. Note $lc(\Gamma) = 1$.

3 Set $r_i = 1 + (\deg_{x_i} \check{A} \deg_{x_0} \check{B} + \deg_{x_i} \check{B} \deg_{x_0} \check{A})$ for $1 \leq i < n$.

4 Let $Y = (y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{n-1}})$.
Set $K_r(\check{A}) = \check{A}(x, Y)$, $K_r(\check{B}) = \check{B}(x, Y)$ and $K_r(\Gamma) = \Gamma(Y)$.
Set $dy = \min(\deg_y K_r(\check{A}), \deg_y K_r(\check{B})) + \deg_y K_r(\Gamma)$.

5 Construct the set S of smooth primes according to Theorem 8.10 with $X = 4$, $Y = 4$. Therefore Prob[p is good and E are all good] $\geq (1 - \frac{1}{4})^2 = \frac{9}{16} > \frac{1}{2}$.

6 Set $\widehat{H} = 0, M_d = 1, R = Fail, d_0 = \min(\deg_{x_0} \check{A}, \deg_{x_0} \check{B})$.

LOOP: // Invariant: $d_0 \geq \deg_{x_0} H = \deg_{x_0} G$.

7 Call PGCD($K_r(\check{A})$, $K_r(\check{B})$, $K_r(\Gamma)$, S, τ, M_d).
If PGCD outputs Fail then **goto** LOOP.
Let p and $\widehat{H}p = \sum_{i=0}^{dx} \widehat{h}_i(y) x^i$ be the output of PGCD where $\widehat{h}_i(y) \in \mathbb{Q}(\alpha)[y]$.

8 If $dx > d_0$ then either p is unlucky or all evaluation points were unlucky so **goto** LOOP.

9 If $dx < d_0$ then either this is the first image or all previous images in $\widehat{H}$ were unlucky so set $d_0 = d_x$, $\widehat{H} = \widehat{H}p$, $M_d = p$ and **goto** LOOP.

Chinese Remaindering and Rational Reconstruction:

10 If R =Fail then Set $Hold = \widehat{H}$. Solve $\{\widehat{H} \equiv Hold \mod M_d$ and $\widehat{H} \equiv \widehat{H}p \mod p\}$ for $\widehat{H}$. Set $M_d = M_d \times p$. Apply rational reconstruction to obtain $\widehat{R} \in \mathbb{Q}(\alpha)[x, y]$ from $\widehat{H}$ mod M_d. If rational reconstruction fails then **goto** LOOP else $R = \widehat{R}$.

11 If $\widehat{H}p \neq R \mod p$ then **goto** LOOP.

Termination.

12 Set $\widetilde{R} = K_r^{-1}R(x, y)$ and let $\widetilde{R} = \sum_{i=0}^{d_0} \widetilde{c}_i x_0^i$ where $\widetilde{c}_i \in \mathbb{Q}(\alpha)[x_1, x_2, \ldots, x_n]$.

13 Set $\widehat{G} = \widetilde{R}/\gcd(\widetilde{c}_0, \widetilde{c}_1, \ldots, \widetilde{c}_{d_0})$ ($\widehat{G}$ is the primitive part of $\widetilde{R}$ and $lc(\widehat{G}) = 1$).

14 If $\widehat{G}|A$ and $\widehat{G}|B$ then **output** $\widehat{G}$.

15 Set $R =$ Fail and **goto** LOOP.

Algorithm PGCD$_\alpha$($K_r(\check{A})$, $K_r(\check{B})$, $\check{m}(z)$, $K_r(\Gamma)$, S, τ, M)

Inputs $K_r(\check{A}), K_r(\check{B}) \in \mathbb{Q}(\alpha)[x, y]$, $K_r(\Gamma) \in \mathbb{Q}(\alpha)[y]$, $\check{m}(z) \in \mathbb{Z}[z]$. S a set of smooth primes, a term bound $\tau \geq \max \#h_i$ and M a positive integer.

Output With probability $\geq \frac{1}{2}$ a prime p and polynomial $Hp \in \mathbb{Z}_p[x, y]$ satisfying $Hp = K(H) \bmod p$ and p does not divide M. Or Fail.

1 Pick a prime p uniformly at random from S that is not bad and does not divide M.

2 Pick a shift $s \in \mathbb{Z}_p^*$ uniformly at random and any generator ω for $\mathbb{Z}_p^*$.

Compute-and-scale-images:

3 For j from 0 to $2\tau - 1$ do

4 Compute $a_j = K_r(\check{A})(x, \omega^{s+j}) \bmod p$ and $b_j = K_r(\check{B})(x, \omega^{s+j}) \bmod p$.

5 If $\deg_x a_j < \deg_x K_r(\check{A})$ or $\deg_x b_j < \deg_x K_r(\check{B})$ then **output** Fail (ω^{s+j} is a bad evaluation point.)

6 Compute $g_j = \gcd(a_j, b_j) \in \mathbb{Z}_p(\alpha)[x]$ using the monic subresultant PRS algorithm, see Figure 8.2. If $g_j =$ Fail then **output** Fail else set $g_j = K(\Gamma)(\omega^{s+j}) \times g_j \bmod p$.

End for loop.

7 Set $d_0 = \deg g_0(x)$. If $\deg g_j(x) \neq d_0$ for any $1 \leq j \leq 2\tau - 1$ **output** Fail (unlucky evaluations).

8 Regard α as a variable hence $g_i \in \mathbb{Z}_p[x, \alpha]$. Let $S = \cup_{i=0}^{2\tau-1}\{\text{Support}(g_i)\}$.

Interpolate-coefficients:

9 For $i = 1$ to $|S|$ do

10 Run the Berlekamp-Massey algorithm on the coefficients of $S[i]$ in the images $g_0, g_1, \ldots, g_{2\tau-1}$ to obtain $\lambda_i(z)$ and set $\tau_i = \deg \lambda_i(z)$.

11 Compute the roots m_j of each $\lambda_i(z)$ in $\mathbb{Z}_p$. If the number of distinct roots of $\lambda_i(z)$ is not equal τ_i then **output** Fail (the feedback polynomial is wrong due to undetected unlucky evaluations.)

12 Set $e_k = \log_\alpha m_k$ for $1 \leq k \leq \tau_i$. If $e_k > dy$ for some k then **output** Fail else set $\sigma_i = \{y^{e_1}, y^{e_2}, \ldots y^{e_{\tau_i}}\}$.

13 Solve the τ_i by τ_i shifted transposed Vandermonde system

$$\left\{\sum_{k=1}^{\tau_i}(\omega^{s+j})^{e_k}u_k = \text{coefficient of } S[i] \text{ in } g_j(x,\alpha) \;\; \textbf{for } 0 \leq j < \tau_i\right\}$$

modulo p for u and set $h_i(y) = \sum_{k=1}^{\tau_i} u_k y^{e_k}$. Note: $(\omega^{s+j})^{e_k} = m_k^{s+j}$

End for loop.

14 Set $Hp := \sum_{i=1}^{|S|} h_i(y)S[i]$ and **output** (p, Hp).

MGCD$_\alpha$ is a Las-Vegas algorithm. It always returns the correct result. Step 13 in PGCD$_\alpha$ must have a unique solution due to the choice of the prime. We note that $d_0 \geq \deg_{x_0} \widehat{H} = \deg_{x_0} \widehat{G}$ throughout MGCD$_\alpha$. If we choose $X = 4$ and $Y = 4$, then at least $\frac{3}{4}$ primes in S are good and at least $\frac{3}{4}$ of the possible evaluation sequences are good. Hence PGCD$_\alpha$ outputs a good image of $\widehat{H}$ with probability at least $\frac{1}{2}$. Eventually, $d_0 = \deg_{x_0} \widehat{H} = \deg_{x_0} \widehat{G}$. Once d_0 is correct, all unlucky images can detected. With enough images $\widehat{H}$ of degree d_0, rational number reconstruction determines the correct R. We have implemented MGCD$_\alpha$ in Maple by using the package Algebraic[RecursiveDensePolynomials]. But the package in Maple has a bug which leads to a crash if the prime is large. So we use Michael Monagan's code instead which can be downloaded at

http://www.cecm.sfu.ca/CAG/code/lucas/recden

The Maple file of MGCD$_\alpha$ can be downloaded at

http://www.cecm.sfu.ca/CAG/code/lucas/MGCD.pdf

You need both files to run the examples.

8.8 Rational number reconstruction

Suppose G is the final correct GCD and R is the result from the rational number reconstruction corresponding to G in step 10 of MGCD_α. Let $R = \sum_{i=0}^{d_0} \frac{n_i}{b_i} m_i$ where $n_i, b_i \in \mathbb{Z}$ and m_i is the monomial in α, x, y. Let $N > \max_i\{|n_i|\}$ and $D > \max_i\{|b_i|\}$. If $M_d > 2ND$, then we can use the extended Euclidean algorithm to determine all coefficients $\frac{n_i}{b_i}$ uniquely, see [Wang, 1981] and [Monagan, 2004]. But we don't know R hence we want to express ND in terms of the inputs $A, B, m(z)$. G is a monic factor of A and B, this leads us to use Lemma 8.6. Since Lemma 8.6 only applies to algebraic integers, we have to convert the inputs as we did in Section 8.4. Recall that the main loop in algorithm MGCD_α computes $R = K_r(H) = K_r(\Delta)K_r(G)$ where $\Gamma = \gcd(LC(A), LC(B))$ and $\Delta = \Gamma/LC(G)$.

We follow the notations used in Section 8.4. Let $D_m = den(m(z))$ and $\deg_z m(z) = d_m$. Let $A_c, B_c \in \mathbb{Q}(D_m\alpha)[x_0, \ldots, x_n]$ be the converted inputs and $\check{A}_c, \check{B}_c \in \mathbb{Z}(D_m\alpha)[x_0, \ldots, x_n]$ be the semi-associates of A_c, B_c. Let $G_c = \gcd(A_c, B_c) = \gcd(\check{A}_c, \check{B}_c) \in \mathbb{Q}(D_m\alpha)[x_0, \ldots, x_n]$. Let $s(v)$ denote the minimal polynomial of $D_m\alpha$ which is an algebraic integer. In Section 8.4 we also defined $h_c = \max(|A_c|_{nmax}, |B_c|_{nmax})$ and $d_c = \max(den(A_c), den(B_c))$, then $\max(\|\check{A}_c\|, \|\check{B}_c\|) \le d_c h_c$. Since K_r is invertible for $\check{A}_c, \check{B}_c$,

$$d_c h_c \ge \max(\|K_r(\check{A}_c)\|, \|K_r(\check{B}_c)\|).$$

Recall that $discr(s(v))G_c \in \mathbb{Z}(\alpha)[x_0, \ldots, x_n]$ where $discr(s(v))$ denotes the discriminant of $s(v)$. By Lemma 8.6 we have

$$\|discr(s(v))K_r(G_c)\| \le E_c d_c h_c |discr(s)|$$

where $E_c = e^{(n+1)d} d_m (d_m - 1)^{\frac{d_m-1}{2}} \|s(v)\|_2^{d_m-1} |discr(s)|^{-\frac{1}{2}} \sum_{i=0}^{d_m-1} \|s(v)\|_2^i$. Now let $\Gamma_c = \gcd(LC(\check{A}_c), LC(\check{B}_c))$ and $\Delta_c = \Gamma_c/LC(G_c)$. $K_r(\Delta_c)$ is a monic factor of $K_r(LC(\check{A}_c))$ and $K_r(LC(\check{B}_c))$. Since $\|K_r(LC(\check{A}_c))\| \le \|K_r(\check{A}_c)\|$ and $\|K_r(LC(\check{B}_c))\| \le \|K_r(\check{B}_c)\|$, by Lemma 8.6 again we have

$$\|discr(s(v))K_r(\Delta_c)\| \le E_l d_c h_c |discr(s)|$$

where $E_l = e^{nd} d_m (d_m - 1)^{\frac{d_m-1}{2}} \|s(v)\|_2^{d_m-1} |discr(s)|^{-\frac{1}{2}} \sum_{i=0}^{d_m-1} \|s(v)\|_2^i$.

The rational number reconstruction in MGCD_α computes $R = K_r(H) = K_r(\Delta_c)K_r(G_c)$. First we note that $\|den(K_r(H))K_r(H)\| \ge ND$. If we can find $\mathbf{B}_r$ so that $\mathbf{B}_r \ge 2ND$ then $M_d > \mathbf{B}_r$ guarantees that the rational number reconstruction finds the unique result, provided that all modular images are correct. It is obvious that

$$\|den(K_r(H))K_r(H)\| \le \|discr(s(v))K_r(G_c) \cdot discr(s(v))K_r(\Delta_c) \mod s(v)\|$$

because $den(K_r(H))$ computes the integer LCM of the denominators of all coefficients in $K_r(H)$ and it is the smallest positive integer which can clear the denominator of $K_r(H)$.

Note that reducing $s(v)$ does not produce fractions because it is a minimal polynomial of an algebraic integer. Since the $\deg_v K_r(\Delta_c) \leq d_m - 1$ and $\deg_y K_r(\Delta_c) \leq (2d^2+1)^n$, we have $\#K_r(\Delta_c) \leq (d_m)(2d^2+1)^n$. By Lemma 8.4,

$$\begin{aligned}&\|discr(s)K_r(G_c) \cdot discr(s)K_r(\Delta_c)\| \\ \leq& d_m(2d^2+1)^n \max(\|discr(s)K_r(G_c)\|, \|discr(s)K_r(\Delta_c)\|).\end{aligned}$$

Since $E_c > E_l$, we have

$$\|discr(s)K_r(G_c) \cdot discr(s)K_r(\Delta_c)\| \leq d_m(2d^2+1)^n(E_c d_c h_c |discr(s)|)^2.$$

Since $\deg_v K_r(G_c)K_r(\Delta_c) \leq d_m - 1 + d_m - 1 = 2d_m - 2$, by Lemma 8.5, we have

$$\begin{aligned}&\|discr(s)K_r(G_c) \cdot discr(s)K_r(\Delta_c) \mod s(v)\| \\ \leq& d_m(2d^2+1)^n(E_c d_c h_c |discr(s)|)^2(1+\|s(v)\|)^{2d_m-2-d_m+1}.\end{aligned}$$

Therefore $\|den(K_r(H))K_r(H)\| \leq d_m(2d^2+1)^n(E_c d_c h_c |discr(s)|)^2(1+\|s(v)\|)^{d_m-1}$. Let

$$\mathbf{B}_r = \mathbf{2}d_m(2d^2+1)^n(E_c d_c h_c |discr(s)|)^2(1+\|s(v)\|)^{d_m-1}.$$

Since $\|den(K_r(H))K_r(H)\| \geq ND$, $\mathbf{B}_r \geq 2ND$. If $M_d > \mathbf{B}_r$ then the rational reconstruction algorithm can determine the result, provided that all modular images are correct. If we compare $\mathbf{B}_r$ with $\mathbf{B}_{up}$ in Section 8.4, we found that $\log_{pmin} \mathbf{B}_r$ is smaller than $\log_{pmin} \mathbf{B}_{up}$. Therefore the set S constructed in the previous section has enough number of primes for the rational number reconstruction.

Let $N_p = \lceil \log_{pmin} \mathbf{B}_r \rceil$ where $pmin = \min S$. The rational number reconstruction needs at most N_p primes to terminate. We also notice that at most $\log_{pmin} ND$ good primes in S can determine the complete support of R.

Theorem 8.11. *$N_p = \lceil \log_{pmin} \mathbf{B}_r \rceil$ where $pmin = \min S$ and let Z be the number of calls that Algorithm MGCD$_\alpha$ makes to PGCD$_\alpha$. Then $E[Z] \leq 2(N_p + 1)$.*

Proof. Since we choose $X = 4$ and $Y = 4$ in MGCD$_\alpha$ to construct the set S according to Theorem 8.10, the probability that PGCD$_\alpha$ outputs a good image of $K_r(H)$ is at least $\frac{1}{2}$. Since we need N_p images of $K_r(H)$ to reconstruct the coefficients and one more to stabilize, $E[Z] \leq 2(N_p + 1)$. □

Chapter 9

Conclusion

In this thesis, we have developed efficient modular polynomial GCD algorithms for multivariate polynomials over $\mathbb{Z}$. They are based on Ben-Or/Tiwari sparse interpolation. Compared with Zippel's GCD algorithm which uses $O(ndt)$ evaluation points, our GCD algorithms use $O(t)$ evaluation points. The discrete logarithm method helps us to reduce the size of primes from about $O(n \log d)$ to $O(D \log n \log \log n)$ where d bounds the degree of each variable in the target GCD and D is the total degree of the target GCD. See Table 2.1. Since t is unknown, for a given prime, we may run out of evaluation points. This leads us to consider Kronecker substitution. A Kronecker substitution reduces a multivariate GCD problem to a bivariate problem, and therefore we just need to interpolate one variable and $p - 1 > 2t$ is normally not a problem. Besides, the evaluation points can be chosen randomly in $\mathbb{Z}_p$. Generally, the Ben-Or/Tiwari sparse interpolation with the discrete logarithm method developed in this thesis can be applied to any application of sparse multivariate polynomial interpolation.

The timing results are very good. See Table 7.1 and Table 7.2. For our benchmark problem where $\#G \approx 10^4$, $\#A \approx 10^6$, $\#B \approx 10^6$ and $t = 1198$. Maple takes 22,111 seconds, Magma takes 1,611 seconds. Our new algorithm takes 48.17 seconds on 1 core and 4.67 seconds on 16 cores. If we compute bivariate GCD images in the base case, then our new algorithm takes 7.614 seconds on 1 core and 0.685 seconds on 16 cores.

For polynomial inputs with more variables and higher degrees, algorithm PGCD probably requires primes p larger than 127 bits. In this case our GCD algorithm still works but requires multi-precision arithmetic. In order to use machine arithmetic to solve a larger GCD problem, our first thought is to further reduce the size of primes for a given problem. In other words, we try to use 127 bits primes to solve GCD problems with inputs as large as possible.

One possible approach is Murao and Fujise's method [Murao and Fujise, 1996] which uses primes smaller than $\deg_y K_r(G)(x, y)$. But Chinese remaindering is required to recover the exponents of the monomials in $K_r(G)$. Their method requires k small primes to get one modular image (one call to our PGCD), so $2tk$ evaluation points are needed. But if all small

primes are 63 bits, then using k 63 bits primes might be better than using one giant prime k times longer. However, with small primes we have to use Chinese remaindering to combine the exponent images converted from the roots of the feedback polynomials *combinatorially*. Murao and Fujise figured out a way to determine the correct combinations but it is still possible that matches are not unique. Therefore further examination is required.

Another possible approach is to compress Kronecker substitution. See the coefficient ratio method in van der Hoven in [van der Hoven and Lecerf, 2015]. The direct consequence of using a "smaller" Kronecker substitution is to reduce the size of primes.

Experiments showed that evaluations dominate the cost of our GCD algorithm, and therefore reducing t is another way to speed up our algorithm. As we saw in Section 6.3, computing bivariate GCD images in the base case effectively reduces t. Hence t can be further reduced by computing the GCD images in three variables in the base case, provided there is a fast modular GCD algorithm for three variables.

Our polynomial GCD algorithm over $\mathbb{Q}(\alpha)$ can be modified to a practical version by the same approach used in Chapter 5. Let $\mathbb{Q}(\alpha_1, \ldots, \alpha_k)$ be a number field with multiple extensions $\alpha_1, \ldots, \alpha_k$ over $\mathbb{Q}$. For $A, B \in \mathbb{Q}(\alpha_1, \ldots, \alpha_k)[x_0, x_1, \ldots, x_n]$, we can also generalize our algorithm to compute $G = \gcd(A, B)$. The only modification needed is to make the univariate GCD computation in the base case have the capacity to handle coefficients in a multiple extension of $\mathbb{Q}$. This has been done by van Hoeij and Monagan [van Hoeij and Monagan, 2002]. Since every algebraic extension of $\mathbb{Q}$ is separable, a primitive element β in $\mathbb{Q}(\alpha_1, \alpha_2, \ldots, \alpha_k)$ can be computed so that $\mathbb{Q}(\beta) = \mathbb{Q}(\alpha_1, \alpha_2, \ldots, \alpha_k)$. Therefore those bounds derived in Chapter 8 should be valid if a proper conversion is made. But, computing a primitive element in $\mathbb{Q}(\alpha_1, \alpha_2, \ldots, \alpha_k)$ might be extremely expensive. See [Abbott et al., 1986].

We encountered several difficulties when we designed those algorithms. For example, the original design did not include Kronecker substitution. We found out that if the factors of $p-1$ are small then we may run out of evaluation points. One easy fix is to increase the size of primes once this problem is detected but we still want to use machine arithmetic. Kronecker substitution solves this problem. At the beginning we did not know whether the Berlekamp-Massey algorithm would work well enough over $\mathbb{Z}_p$ to give us correct feedback polynomials. It took a while to derive Theorem 2.6. The result turns out to be very good. Especially with 63 bit or 127 bit primes, it is very rare to get wrong feedback polynomials. In the number field case, we had no idea how to derive the bound for the number of zero divisors because the monic Euclidean algorithm produces fractions. Fortunately the monic subresultant GCD algorithm over rings saved the day. Although those bounds are very complicated, they work. Our algorithms seem to be less complicated and easier to understand than Zippel based GCD algorithms, but they require many more sub-algorithms. Integrating them together took much effort.

Bibliography

J Abbott, R Bradford, and J Davenport. The Bath algebraic number package. *Proceedings of SYMSAC '86, ACM press*, pages 250–253, 1986.

V Arvind and P Mukhopadhyay. The monomial ideal membership problem and polynomial identity testing. *ISAAC 2007: Algorithms and Computation*, pages 800–811, 2007.

M Ben-Or and P Tiwari. A deterministic algorithm for sparse multivariate polynomial interpolation. *Proc. of STOC '20, ACM Press*, pages 301–309, 1988.

E R Berlekamp. Nonbinary BCH decoding. *Algebraic Coding Theory, New York: McGraw-Hill, chs 7 and 10*, 1968.

E R Berlekamp. Factoring polynomials over large finite fields. *Mathematics of Computation, 24*, pages 713–735, 1970.

J Broida and S Williamson. *A Comprehensive Introduction to Linear Algebra.* Addison-Wesley ISBN 0-201-50065-5, 1989.

W S Brown. On Euclid's algorithm and the computation of polynomial greatest common divisors. *J. ACM. 18*, pages 478–504, 1971.

W S Brown and J F Traub. On Euclid's algorithm and the theory of subresultants. *J. ACM* **18**. *4*, pages 505–514, 1971.

B W Char, K O Geddes, and G H Gonnet. Gcdheu: Heuristic polynomial GCD algorithm based on integer GCD computation. *J. Symbolic Comp. 7:1. Elsevier*, pages 31–48, 1989.

G E Collins. Subresultants and reduced polynomial remainder sequences. *J. ACM* **14***:1, ACM Press*, pages 128–142, 1967.

D Cox, J Little, and D O'Shea. *Ideals, Varieties and Algorithms.* Springer-Verlag, 1991.

J de Kleine, M Monagan, and A Wittkopf. Algorithms for the non-monic case of the sparse modular GCD algorithm. *Proceedings of ISSAC 2005, ACM Press*, pages 124–131, 2005.

Baron Gaspard Clair Françis Marie Riche de Prony. Essai expérimental et analytique sur les lois de la dilatabilité des fluides élastique et sur celles de la force expansive de la vapeur de léau et de la vapeur de lálkool, á différentes températures. *J. de lÉcole Polytechnique*, pages 24–76, 1795.

J L Dornstetter. On the equivalence between Berlekamps and Euclids algorithm. *IEEE Trans. Inform. Theory, vol. IT-33, no 3*, pages 428–431, 1987.

W Lee E Kaltofen and A Lobo. Early termination in Ben-or/Tiwari sparse interpolation and a hybrid of Zippel's algorithmm. *Proceedings of ISSAC 2000, ACM Press*, pages 192–201, 2000.

M J Encarnacion. Computing GCDs of polynomials over algebraic number fields. *J. Symbolic Computation 20*, pages 299–313, 1995.

K O Geddes, S R Czapor, and G Labahn. *Algorithms for Computer Algebra*. Kluwer, 1992.

A Gelfond. *Transcendental and Algebraic Numbers*. GITTL, Moscow. English translation by Leo F. Boron, Dover, New York, 1960.

M Giesbrecht, G Labahn, and W-S Lee. Symbolic-numeric sparse interpolation of multivariate polynomials. *Proceedings of ISSAC 2006*, 2009.

A Goldstein and G Graham. A Hadamard-type bound on the coefficients of a determinant of polynomials. *SIAM Review* **1**, pages 394–395, 1974.

D Grigoriev and M Karpinsk. The matching problem for bipartite graphs with polynomially bounded permanents is in NC. *Proc. 28th IEEE Symposium on the Foundation of Computer Science*, pages 166–172, 1987.

W Habicht. Eine Verallgemeinerung des Sturmschen Wurzelzählverfahrens. *Commun. Math. Helvetici, 21*, pages 99–116, 1948.

W Henkel. Another description of the Berlekamp-Massey algorithm. *Communications IEEE Transactions on, vol. 40*, pages 1557–1561, 1992.

J Hu and M Monagan. A fast parallel sparse polynomial GCD algorithm. *Proceedings of ISSAC 2016, ACM Press*, 2016.

K Imamura and W Yoshida. A simple derivation of the Berlekamp-Massey algorithm and some applications. *IEEE Trans. on Information Theory, 33*, pages 146–150, 1987.

Mahdi Javadi. A new solution to the polynomial GCD normalization problem. *MOCAA M^3 Workshop*, 2008.

Mahdi Javadi and Michael Monagan. Parallel sparse polynomial interpolation over finite fields. *Proceedings of PASCO 2010, ACM Press*, pages 160–168, 2010.

E Jonckheere and C Ma. A simple Hankel interpretation of the Berlekamp-Massey algorithm. *Linear Algebra and its Applications. 125*, pages 65–76, 1989.

E Kaltofen. Fifteen years after DSC and WLSS2 what parallel computations i do today. *Proceedings of PASCO 2010, ACM Press*, pages 10–17, 2010.

E Kaltofen, Y N Lakshman, and J-M Wiley. Modular rational sparse multivariate interpolation algorithm. *Proceedings of ISSAC 1990, ACM Press*, pages 135–139, 1990.

Erich Kaltofen. Sparse Hensel lifting. *Proceedings of EUROCAL 1985*, pages 4–17, 1985.

Erich Kaltofen and Barry Trager. Computing with polynomials given by black boxesfor their evaluations: greatest common divisors, factorization, separation of numerators and denominators. *Proceedings of FOCS 1988, IEEE*, pages 96–105, 1988.

L Langemyr. An asymptotically fast probabilistic algorithm for computing polynomial GCD's over an algebraic number field. *In AAECC-8, volume 508 of Lecture Notes in Computer Science*, pages 222–233, 1990.

L Langemyr and S McCallum. The computation of polynomial GCD's over an algebraic number field. *J. Symbolic Computation 8*, pages 429–228, 1989.

A Lenstra. Factoring multivariate polynomials over algebraic number fields. *SIAM J. Comput. 16 , no. 3*, pages 591–598, 1987.

J Massey. Shift-register synthesis and BCH decoding. *IEEE Trans. on Information Theory, 15*, pages 122–127, 1969.

Niels Möller and Torbjorn Granlund. Improved division by invariant integers. *IEEE Trans. on Computers, 60*, pages 165–175, 2011.

M Monagan. Maximal quotient rational reconstruction: An almost optimal algorithm for rational reconstruction. *Proceedings of ISSAC 2004, ACM Press*, pages 243–249, 2004.

M Monagan and A Wong. Fast parallel multi-point evaluation of sparse polynomials. *Proceedings of PASCO 2017*, 2017.

Michael Monagan and Baris Tuncer. Using sparse interpolation in Hensel lifting. *CASC 2016, Bucharest, Romania*, 2016.

Joel Moses and David Y Y Yun. The EZ GCD algorithm. *Proceedings of ACM 1973, ACM Press*, pages 159–166, 1973.

Gary Mullen and Daniel Panario. *Handbook of Finite Fields.* CRC Press, 2013.

Hirokazu Murao and Tetsuro Fujise. Modular algorithm for sparse multivariate polynomial interpolation and its parallel implementation. *J. Symb. Cmpt. 21*, pages 377–396, 1996.

S Pohlig and M Hellman. An improved algorithm for computing logarithms over GF(p) and its cryptographic significance. *IEEE Trans. on Information Theory. 24*, pages 106–110, 1978.

Michael Rabin. Probabilistic algorithms in finite fields. *SIAM J. Comput, 9*, pages 273–280, 1979.

M O Rayes, P S Wang, and K Weber. Parallelization fo the sparse modular GCD algorithm for multivariate polynomials on shared memory processors. *Proceedings of ISSAC 1994, ACM Press*, pages 66–73, 1994.

C Rubald. Algorithms for polynomials over a real algebraic number field. *Phd thesis, University of Wisconsin, Madison*, 1974.

Massimiliano Sala. *Gröbner Bases, Coding, and Cryptography.* Berlin : Springer; Linz, Austria : RISC, 2009.

Jack Schwartz. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM, 27*, pages 701–717, 1980.

Y Sugiyama, M Kashara, S Hirashawa, and T Namekawa. A method for solving key equation for decoding Goppa codes. *Information and Control, 27*, pages 87–99, 1975.

J Sylvester. On a theory of the syzygetic relations of two rational integral functions, comprising an application to the theory of sturm's functions, and that of the greatest algebraical common measure. *Philosphophical Transactions 143*, pages 407–548, 1853.

M Tang, B Li, and Z Zeng. Computing sparse gcd of multivariate polynomials via polynomial interpolation. *Journal of Systems Science and Complexity, volume 31, issue 2*, pages 552–568, 2018.

P Tauvel and R Yu. *Lie Algebras and Algebraic Groups*. Springer Science and Business Media, 2005.

Joris van der Hoven and Grégoire Lecerf. On the bit complexity of sparse polynomial multiplication. *J. Symb. Cmpt.* **50**, pages 227–254, 2013.

Joris van der Hoven and Grégoire Lecerf. Sparse polynomial interpolation in practice. *ACM Communications in Computer Algebra 48*, pages 187–191, 2015.

M van Hoeij and M Monagan. A modular GCD algorithm over number fields presented with multiple field extensions. *Proceedings of ISSAC 2002, ACM Press*, pages 109–116, 2002.

H van Tilborg and S Jajodia. *Encyclopedia of Cryptography and Security*. 2nd edition, Springer, 2011.

J von zur Gathen and J Gerhard. *Modern Computer Algebra*. Cambridge University Press, UK, 1999.

Paul Wang. An improved multivariate polynomial factoring algorithm. *Mathematics of Computation, 32, 144*, pages 1215–1231, 1978.

Paul Wang. The EEZ-GCD algorithm. *ACM SIGSAM Bulletin, 14, 2*, pages 50–60, 1980.

Paul Wang. A p-adic algorithm for univariate partial fractions. *SYMSAC 81: Proceedings of the fourth ACM symposium on Symbolic and algebraic computation*, 1981.

P Weinberger and L Rothschild. Factoring polynomials over algebraic number fields. *ACM Transactions on Mathematical Software (TOMS): Volume 2 Issue 4*, 1976.

Allan Wittkopf. Algorithms and implementations for differential elimination. *Phd thesis, Simon Fraser University, Canada*, 2004.

Richard Zippel. Probabilistic algorithms for sparse polynomials. *PhD thesis, Massachusetts Inst. of Technology, Cambridge, USA*, 1979a.

Richard Zippel. Probabilistic algorithms for sparse polynomials. *Proceedings of EUROSAM 1979*, pages 216–226, 1979b.

Richard Zippel. Interpolating polynomials from their values. *J. Symb Cmpt. 9*, pages 375–403, 1990.

Appendix A

Code

Maple code for the 6 variable gcd benchmark.

```
r := rand(2^31);
X := [u,v,w,x,y,z];
getpoly := proc(X,t,d) local i,e;
     e := rand(d+1);
     add( r()*mul(x^e(),x=X), i=1..t );
end:

infolevel[gcd] := 3; # to see output from Zippel's algorithm

for d in [5,10,20,50,100] do
  s := 100; t := 100*d;
  g := add(x^d,x=X) + r() + getpoly(X,t-7,d-1);
  abar := getpoly(X,s-1,d) + r();  a := expand(g*abar);
  bbar := getpoly(X,s-1,d) + r(); b := expand(g*bbar);
  st := time(); h := gcd(a,b); gcdtime := time()-st;
  printf("d=%d  time=%8.3f\n",d,gcdtime);
end do:
```

Magma code for the 6 variable gcd benchmark.

```
p := 2^31;
Z := IntegerRing();
P<u,v,w,x,y,z> := PolynomialRing(Z,6);

randpoly := function(d,t)
M := [ u^Random(0,d)*v^Random(0,d)*w^Random(0,d)
      *x^Random(0,d)*y^Random(0,d)*z^Random(0,d) : i in [1..t] ];
C := [ Random(p) : i in [1..t] ];
g := Polynomial(C,M);
return g;
end function;
```

```
for d in [5,10,20,50] do
  s := 100; t := 100*d;
  g := u^d+v^d+w^d+x^d+y^d+z^d + randpoly(d-1,t-7) + Random(p);
  abar := randpoly(d,s-1) + Random(p);  a := g*abar;
  bbar := randpoly(d,s-1) + Random(p);  b := g*bbar;
  d;  time h := Gcd(a,b);
end for;
```