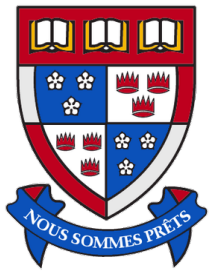
PhD Thesis Proposal: Computing Gcds of Multivariate Black-Box Polynomials

Garrett Paluck

Department of Mathematics, Simon Fraser University, British Columbia

July 26, 2024



Motivation for Thesis

Let $a, b \in \mathbb{Z}[x_1, ..., x_n]$ be represented by the black-boxes $\mathbf{B}_a$ and $\mathbf{B}_b$.

How can we calculate $g = \gcd(a, b)$? Using $\mathbf{B}_a$ and $\mathbf{B}_b$?

History:

- Kalotfen and Trager (1990): Created the first black-box gcd algorithm
- Kaltofen and Diaz (1995): Created an algorithm which constructs a black-box representation of g, namely $\mathbf{B}_g$, from $\mathbf{B}_a$ and $\mathbf{B}_b$.
  - This algorithm has been implemented in Maple.
- Lecerf and Van Der Hoeven (2023): Created a new black-box gcd algorithm for sparse polynomials using projective coordinates.

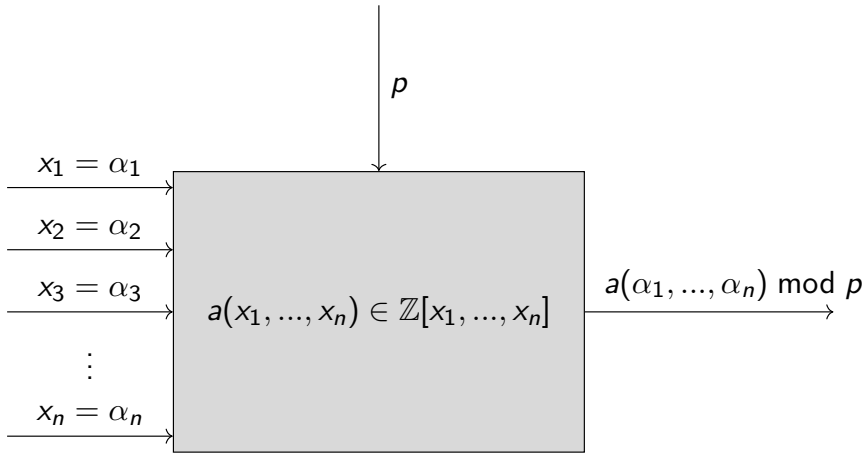
Sparse Polynomial and Modular Black-box definitions

We are interested in calculating the gcd of sparse polynomials. Let

$$a = \sum_{j=1}^{t} a_j M_j(x_1, ..., x_n) = \sum_{j=1}^{t} a_j \cdot x_1^{e_{1,j}} x_2^{e_{2,j}} \dots x_n^{e_{n,j}}$$

where the coefficients $a_j \neq 0$ are integers and $(e_{1,j}, e_{2,j}, ..., e_{n,j}) \in \mathbb{N}^n$

A modular black-box representation of $a \in \mathbb{Z}[x_1, ..., x_n]$ is a computer program $\mathbf{B}_a$ that takes as input a point $\alpha \in \mathbb{Z}^n$ and a prime p and outputs $a(\alpha) \bmod p$.



Outline

Our thesis is broken down into 4 projects:

1. A bivariate Hensel lifting algorithm (BHL)
2. An optimized fast Euclidean algorithm (OFEA)
3. MHLBBPGCD a BB gcd algorithm using multivariate Hensel lifting
4. SEPPGCD a BB gcd algorithm using rational function interpolation

The MHLBBPGCD and SEPPGCD algorithms compute $\text{pp}(g)$.

Example:

$$g = 6x^2 - 4y^2 \qquad \text{pp}(g) = 3x^2 - 2y^2$$

The BHL and OFEA algorithms aid in calculating $\text{pp}(g)$.

The Bivariate Hensel Lifting Algorithm

Let's assume we have $A = \prod_{i=1}^{r} h_i(x,y) \in \mathbb{Z}_p[x,y]$ and $f_i^{(0)} = h_i(x,\alpha)$ for some $\alpha \in \mathbb{Z}_p$. α is selected such that $\gcd(f_i^{(0)}, f_j^{(0)}) = 1$ for $i \neq j$.

The goal of Hensel lifting is to lift the initial factors $h_i(x,\alpha) \to h_i(x,y)$ to recover the factorization of A in $\mathbb{Z}_p[x,y]$.

Hensel lifting in $\mathbb{Z}_p[x,y]$ - r factor case

Input: $A \in \mathbb{Z}_p[x,y]$, $f_1^{(0)}, f_2^{(0)}, .., f_r^{(0)} \in \mathbb{Z}_p[x]$, $\alpha \in \mathbb{Z}_p$, and $m \in \mathbb{N}$ s.t.
(i) $A(x,y) \equiv \prod_{i=1}^{r} f_i^{(0)} \mod (y-\alpha)$, and
(ii) $\gcd(f_i^{(0)}, f_j^{(0)}) = 1$ for $i \neq j$
Output: $h_1, ..., h_r \in \mathbb{Z}_p[x,y]$ satisfying
(i) $A \equiv \prod_{i=1}^{r} h_i(x,y) \mod (y-\alpha)^m$ and
(ii) $h_i(x,y) \equiv f_i^{(0)} \mod (y-\alpha)$ for $1 \leq i \leq r$

Note: In Hensel lifting we want to stop lifting if $A - \prod_{i=1}^{r} h_i = 0$.

The Bivariate Hensel Lifting Algorithm: Main Result

Let $A = \prod_{i=1}^{r} h_i$ in $\mathbb{Z}_p[x, y]$ and $\alpha \in \mathbb{Z}_p$. Let $d_x = \deg(A, x)$ and $d_y = \deg(A, y)$.

In [MP19] we created the bivariate Hensel lifting algorithm for lifting $r \geq 2$ polynomials. It required that A be **monic** in x and used $O(d_x^2\, d_y + d_x\, d_y^2)$ arithmetic operations in $\mathbb{Z}_p$.

Improvements:

- Our bivariate Hensel lifting algorithm now works for **non-monic** A with the same arithmetic cost.
- We have reduced the space complexity from $O(r\, d_x\, d_y)$ to $O(\log_2 r\, d_x\, d_y)$.

Benchmarks

For $A = \prod_{i=1}^{r} h_i$ for $r = 4$ factors in $\mathbb{Z}_p[x, y]$ where $p = 2^{31} - 1$ and $d = \deg(f_i, x) = \deg(f_i, y)$ for $1 \leq i \leq 4$.

We compare

- Bernardin's $O(r\, d_x^2\, d_y^2)$ algorithm in C
- Our $O(d_x^2 d_y + d_x d_y^2)$ algorithm in C
- Quadratic Hensel lifting $O(\log_2 r\, M(d_x\, d_y))$ in Magma

For $A = \prod_{i=1}^{r} h_i$ for $r = 4$ factors in $\mathbb{Z}_p[x, y]$ where $p = 2^{31} - 1$ and $d = \deg(h_i, x) = \deg(h_i, y)$ for $1 \leq i \leq 4$. We set $d_x = d_y = 4d$.

d_x/d_y	d	LHL (Bernardin) $O(d_x^2 d_y^2)$	LHL (Our cubic result) $O(d_x^2 d_y + d_x d_y^2)$	QHL	
				Timing 1	Timing 2
8	2	0.10ms	0.06ms	1.16ms	3.44ms
16	4	0.22ms	0.18ms	7.48ms	17.82ms
32	8	0.84ms	0.54ms	30.30ms	65.61ms
64	16	4.82ms	2.12ms	125.22ms	247.44ms
128	32	48.25ms	10.28ms	619.90ms	1,225.6ms
256	64	505.64ms	61.18ms	3.16s	6.33s
512	128	6.002s	401.14ms	17.58s	35.21s
1024	256	76.70s	3.41s (0.12gb)	97.46s	204.27s
2048	512	1,074s	25.17s (0.46gb)	553.85s	1,137.71s
4096	1024	15,878s	190.13s (1.83gb)	2,975s	5,599.3s
8192	2048	NA	1,462s (7.32gb)	15,583s	>32gb
16384	4096	NA	12,614.7s (29.3gb)	NA	NA

Optimizing the Fast Euclidean Algorithm in $\mathbb{Z}_p[x]$

I have implemented an optimized version of the Fast Euclidean Algorithm in C using the Half Gcd algorithm from [GJ13] to calculate the gcd of two polynomials over $\mathbb{Z}_p[x]$ for a Fourier prime p.

	Complexity
Euclidean algorithm	$O(d^2)$
Fast Euclidean algorithm	$O(M(d)\log d)$

The work is motivated by Huang and Monagan in [HM24] who reduce the calculation of the sparse gcd of two polynomials in $\mathbb{Z}[x_1, ..., x_n]$ to many gcd calculations in $\mathbb{Z}_p[x_1]$ of high degree.

Optimizations:

- minimizing the total number of uses of the FFT algorithm.
- Using the extended Euclidean algorithm for low degree recursive calls to the Half Gcd algorithm.

Minimizing the total number of the FFT calls

Let $f \in F[x]$ where $\deg(f) < n = 2^k$ and let $\omega \in F$ be a primitive nth root of unity. We define the Discrete Fourier Transform as:

$$\mathrm{D}_\omega : f \mapsto \left(f(1), f(\omega), f(\omega^2), ..., f(\omega^{n-1})\right)$$

Let $A, B, C \in F[x]^{2\times 2}$ s.t.

$$A = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}, B = \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}, \text{ and } C = \begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix}.$$

We want to calculate $C = A \times B$ using FFT multiplication. For simplicity, let's assume the degree of every polynomial in A, B and C is less than $n = 2^k$.

Minimizing the total number of the FFT calls

We use the calculation of polynomial c_{11} to illustrate the optimization

$$\begin{bmatrix} c_{11} & c_{12} \\ c_{21} & c_{22} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \times \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

Inefficient Method:

$$c_{11} = \frac{1}{n}\mathrm{D}_{\omega}^{-1}(\mathrm{D}_{\omega}(a_{11}) \cdot \mathrm{D}_{\omega}(b_{11})) + \frac{1}{n}\mathrm{D}_{\omega}^{-1}(\mathrm{D}_{\omega}(a_{12}) \cdot \mathrm{D}_{\omega}(b_{21}))$$

Each polynomial in C requires 6 FFT calls (24 total)

Efficient Method:

Precompute D_{ω} of each polynomial in A and B once (8 total)

Since

$$\mathrm{D}_{\omega}(f + g) = \mathrm{D}_{\omega}(f) + \mathrm{D}_{\omega}(g)$$

$$c_{11} = \frac{1}{n}\mathrm{D}_{\omega}^{-1}(\mathrm{D}_{\omega}(a_{11}) \cdot \mathrm{D}_{\omega}(b_{11}) + \mathrm{D}_{\omega}(a_{12}) \cdot \mathrm{D}_{\omega}(b_{21}))$$

This reduces the the uses of D_{ω}^{-1} by 4 (4 total)

Thus, we compute C using 12 total FFT calls

Benchmarks 1 (32 pit prime)

We calculated $\gcd(a, b)$ for two randomly generated polynomials $a, b \in \mathbb{Z}_p[x]$ for $p = 2^{28} + 2^{21} + 1$.

deg a/deg b	Our algorithm	Maple Gcdex	Maple GCD	Euc. Alg. in C	Magma
$2^6 - 1/2^6 - 2$	0.0002	0.0001	0.0001	0.0002	0.0002
$2^7 - 1/2^7 - 2$	0.0004	0.0003	0.0001	0.0004	0.0003
$2^8 - 1/2^8 - 2$	0.0009	0.0010	0.0004	0.0009	0.0003
$2^9 - 1/2^9 - 2$	0.0022	0.0038	0.0016	0.0024	0.0012
$2^{10} - 1/2^{10} - 2$	0.0051	0.0140	0.0061	0.0071	0.0032
$2^{11} - 1/2^{11} - 2$	0.0122	0.0544	0.0240	0.0243	0.0079
$2^{12} - 1/2^{12} - 2$	0.029	0.245	0.092	0.081	0.0195
$2^{13} - 1/2^{13} - 2$	0.068	0.965	0.365	0.326	0.049
$2^{14} - 1/2^{14} - 2$	0.156	3.834	1.454	1.365	0.120
$2^{15} - 1/2^{15} - 2$	0.349	15.176	5.823	5.237	0.320
$2^{16} - 1/2^{16} - 1$	0.787	60.165	23.289	21.149	0.800
$2^{17} - 1/2^{17} - 1$	1.764	239.876	93.172	82.512	1.960
$2^{18} - 1/2^{18} - 1$	3.967	956.839	373.092	329.318	4.700
$2^{19} - 1/2^{19} - 1$	8.838	-	-	-	11.100
$2^{20} - 1/2^{20} - 1$	19.403	-	-	-	26.100

Table: $\gcd(a, b)$ timings for $\mathbb{Z}_p[x]$ for $p = 2^{28} + 2^{21} + 1$

Benchmarks 2 (64 pit prime)

We calculated $\gcd(a, b)$ for two randomly generated polynomials $a, b \in \mathbb{Z}_p[x]$ for $p = 2^{60} + 2^{33} + 1$.

deg a/deg b	Our algorithm	Maple Gcdex	Maple GCD	Euc. Alg. in C	Magma
$2^6 - 1/2^6 - 2$	0.0002	0.0010	0.0004	0.0003	0.0008
$2^7 - 1/2^7 - 2$	0.0005	0.0030	0.0015	0.0005	0.0013
$2^8 - 1/2^8 - 2$	0.0012	0.0111	0.0056	0.0010	0.0049
$2^9 - 1/2^9 - 2$	0.0027	0.0426	0.0220	0.0025	0.0175
$2^{10} - 1/2^{10} - 2$	0.0066	0.1666	0.0868	0.0079	0.0467
$2^{11} - 1/2^{11} - 2$	0.0154	0.6538	0.3443	0.0263	0.1150
$2^{12} - 1/2^{12} - 2$	0.0355	2.5894	1.371	0.0945	0.2714
$2^{13} - 1/2^{13} - 2$	0.082	10.295	5.475	0.367	0.630
$2^{14} - 1/2^{14} - 2$	0.185	40.229	21.874	1.441	1.480
$2^{15} - 1/2^{15} - 2$	0.416	163.967	87.424	5.572	3.190
$2^{16} - 1/2^{16} - 1$	0.934	654.904	348.915	22.153	7.230
$2^{17} - 1/2^{17} - 1$	2.082	2632.626	1406.817	87.748	16.320
$2^{18} - 1/2^{18} - 1$	4.656	10545.097	5619.839	349.733	36.900
$2^{19} - 1/2^{19} - 1$	9.744	-	-	-	83.290
$2^{20} - 1/2^{20} - 1$	21.478	-	-	-	197.530

Table: $\gcd(a, b)$ timings for $\mathbb{Z}_p[x]$ for $p = 2^{60} + 2^{33} + 1$

The MHLBBPGCD and BBMGCD algorithms

Let $a, b \in \mathbb{Z}[x_1, ..., x_n]$ be represented by the black-boxes $\mathbf{B}_a$ and $\mathbf{B}_b$ and $g = \text{pp}(\gcd(a, b))$. Our first method to calculate $g \in \mathbb{Z}[x_1, ..., x_n]$ uses the MHLBBPGCD and BBMGCD algorithms.

The MHLBBPGCD algorithm calculates $g \bmod p \in \mathbb{Z}_p[x_1, ..., x_n]$ for a prechosen prime p using the CMBBSHL algorithm [Che24] for multivariate Hensel lifting.

The BBMGCD algorithm calls the MHLBBPGCD to calculate $g \bmod p \in \mathbb{Z}_p[x_1, ..., x_n]$ for various primes p. It then uses Chinese remaindering and rational number reconstruction to compute g.

MHLBBPGCD Algorithm Outline I

The MHLBBPGCD algorithm calculates the square-free factorization of g. Consider the following square-free factorization of g over $\mathbb{Z}[x_2, ..., x_n][x_1]$

$$g = h \prod_{i=1}^{r} f_i^i$$

where

(i) $\deg(f_i, x_1) > 0$,

(ii) f_i is primitive and square-free in $\mathbb{Z}[x_2, \ldots, x_n][x_1]$,

(iii) $h \in \mathbb{Z}[x_2, \ldots, x_n]$ (h is the content of g) and

(iv) $\gcd(f_i, f_j) = 1$ for $i \neq j$.

MHLBBPGCD Algorithm Outline II

Step 1 Pick $\alpha = (\alpha_2, ..., \alpha_n)$ from $[0, p)^{n-1}$ at random.

Step 2 Interpolate $a_1 = a(x_1, \alpha_2, ..., \alpha_n)$ and $b_1 = b(x_1, \alpha_2, ..., \alpha_n)$ in $\mathbb{Z}_p[x_1]$ by probing the modular black-boxes $\mathbf{B}_a$ and $\mathbf{B}_b$.

Step 3 Compute $h_1 = \gcd(a_1, b_1)$ over $\mathbb{Z}_p[x_1]$ and then compute the square-free factorization $h_1 = \prod_{i=1}^{s} g_i^i$. For most choices of α and p we will have $r = s$ and $g_i \sim f_i(x_1, \alpha)$ in $\mathbb{Z}_p[x_1]$ for $1 \leq i \leq r$.

Step 4 Use a modified CMBBSHL algorithm to lift the square-free part of h_1, $\mathrm{sqf}(h_1) = \prod_{i=1}^{s} g_i$, to construct $\mathrm{sqf}(\mathrm{pp}(g)) = \prod_{i=1}^{s} f_i$ in $\mathbb{Z}_p[x_1, ..., x_n]$. Using the multiplicities from Step 3, we obtain $\mathrm{pp}(g) = \prod_{i=1}^{s} f_i^i$.

Step 5 If $n > 1$ then let $f = \prod_{i=1}^{s} f_i^i$. Construct a new modular black-boxes $\mathbf{B}_c, \mathbf{B}_d : (\mathbb{Z}^{n-1}, p) \to \mathbb{Z}_p$ such that for $\beta \in \mathbb{Z}_p$ and $\gamma \in \mathbb{Z}_p^{n-1}$,
$\mathbf{B}_c(\gamma, p)$ computes $a(\beta, \gamma)/f(\beta, \gamma)$ and
$\mathbf{B}_d(\gamma, p)$ computes $b(\beta, \gamma)/f(\beta, \gamma)$.
Now compute $h \in \mathbb{Z}_p[x_1, ..., x_n]$ recursively using $\mathbf{B}_c$ and $\mathbf{B}_d$.

BBMGCD Algorithm Outline

The BBMGCD algorithm computes g using the MHLBBPGCD algorithm.

Step 1 Set $i = 1$.

Step 2 Pick a new random large prime p_i.

Step 3 Compute $g_{p_i} = g \bmod p_i$ using the MHLBBPGCD algorithm and make it monic in lex order with $x_1 > x_2 > \ldots > x_n$.

Step 4 Find $g^* \in \mathbb{Z}_M[x_1, \ldots, x_n]$ using Chinese remaindering where $M = \prod_{j=1}^{i} p_i$ such that $g^* \equiv g_{p_j} \bmod p_j$ for $1 \leq j \leq i$.

Step 5 Use rational number reconstruction to find a monic $\hat{g} \in \mathbb{Q}[x_1, \ldots, x_n]$ s.t. $\hat{g} \equiv g^* \bmod M$ or $\hat{g} = \text{FAIL}$.

Step 6 Increment i and repeat Steps 2-4. Stop when $\hat{g} \neq \text{FAIL}$ and you got the same $\hat{g}$ for two consecutive primes.

Step 7 Clear the fractions of $\hat{g}$ so that $\hat{g} \in \mathbb{Z}[x_1, \ldots, x_n]$.

The SEPPGCD algorithm

Let $a, b \in \mathbb{Z}[x_1, ..., x_n]$ be represented by the black-boxes $\mathbf{B}_a$ and $\mathbf{B}_b$ and $g = \mathrm{pp}(\gcd(a, b))$. Our second method to calculate $g \in \mathbb{Z}[x_1, ..., x_n]$ uses the SEPPGCD and BBMGCD algorithms.

The SEPPGCD algorithm calculates g mod $p \in \mathbb{Z}_p[x_1, ..., x_n]$ for a prechosen prime p using a multivariate rational function seperation algorithm [KY07] and Ben-Or/Tiwari sparse interpolation [BOT88].

The BBMGCD algorithm can be modified to calculate g mod p using the SEPPGCD algorithm instead of the MHLBBPGCD algorithm. The rest of the algorithm remains unchanged.

SEPPGCD Algorithm Outline I

Consider the factorizations

$$a = g\bar{a} \text{ and } b = g\bar{b}$$

where $\bar{a}, \bar{b} \in \mathbb{Z}[x_1, ..., x_n]$ are relatively prime.

Dividing a by b results in

$$\frac{a}{b} = \frac{g\bar{a}}{g\bar{b}} = \frac{\bar{a}}{\bar{b}}.$$

Let $\alpha \in \mathbb{Z}_p^n$ be a point. We represent the rational function $R = \bar{a}/\bar{b}$ by a black-box $\mathbf{B}_R : (\mathbb{Z}_p, p) \to \mathbb{Z}_p$ which computes

$$\mathbf{B}_R(\alpha, p) = \mathbf{B}_a(\alpha, p)/\mathbf{B}_b(\alpha, p).$$

SEPPGCD Algorithm Outline II

Let $\beta = (\beta_2, ..., \beta_n) \in \mathbb{Z}_p^{n-1}$ be a random point and $\sigma = (\sigma_1, ..., \sigma_n) \in \mathbb{Z}_p^n$ be a point. We use rational function interpolation to obtain

$$T(x) = \frac{\bar{a}(x, \beta_2(x-\sigma_1)+\sigma_2, \cdots, \beta_n(x-\sigma_1)+\sigma_n)/\Delta}{\bar{b}(x, \beta_2(x-\sigma_1)+\sigma_2, \cdots, \beta_n(x-\sigma_1)+\sigma_n)/\Delta}$$

where $\Delta = \mathrm{LC}(\bar{b}(x, \beta_2(x-\sigma_1)+\sigma_2, \cdots, \beta_n(x-\sigma_1)+\sigma_n)) \in \mathbb{Z}_p \setminus \{0\}$.

Observe

$$T(\sigma_1) = \frac{\bar{a}(\sigma_1, \sigma_2, ..., \sigma_n)/\Delta}{\bar{b}(\sigma_1, \sigma_2, ..., \sigma_n)/\Delta}.$$

We then compute

$$\Delta \cdot g(\sigma) = a(\sigma)/(\bar{a}(\sigma)/\Delta).$$

We use $\sigma = (2^j, 3^j, 5^j, ..., p_n^j)$ for the Ben-Or/Tiwari algorithm.

SEPPGCD Algorithm Outline III

Let $g = \sum_{i=1}^{t} c_i M_i(x_1, ..., x_n)$, $m_i = M_i(2, 3, ..., p_n)$, $\lambda(z) = \prod_{i=1}^{t}(z - m_i)$.

Step 1 Set $T = 1$.

Step 2 Create a local black-box $\mathbf{B}_R : (\mathbb{Z}_p^n, p) \to \mathbb{Z}_p$ which takes as input a prime p and a point $\delta \in \mathbb{Z}_p^n$. It outputs FAIL if $\mathbf{B}_b(\delta, p) = 0$ and $\mathbf{B}_a(\delta, p)/\mathbf{B}_b(\delta, p)$ otherwise.

Step 3 Using a multivariate rational function separation algorithm [KY07], compute $\bar{a}(\sigma^{(j)}) \bmod p$ and $\bar{b}(\sigma^{(j)}) \bmod p$ using the point $\sigma^{(j)} = (2^j, 3^j, ..., p_n^j)$ for $0 \leq j < 2T$.

Step 4 Compute $g(\sigma^{(j)}) = a(\sigma^{(j)})/\bar{a}(\sigma^{(j)}) \bmod p$ for $0 \leq j < 2T$.

Step 5 Use the Berlekamp-Massey algorithm to compute $\lambda(z)$.

Step 6 If $\deg(\lambda(z)) = T$ then set $T = 2T$ and repeat steps 3-5. Stop when $\deg(\lambda(z)) < T$.

Step 7 Use Ben-Or/Tiwari sparse interpolation to compute g.

Work Completed

1. The bivariate Hensel lifting algorithm has been implemented in C. We published our results in [MP22] and presented our algorithm at ISSAC 2022 in Lille.
2. I have created a C implementation of our optimized fast Euclidean algorithm.
3. I have created Maple and C implementations of the MHLBBPGCD and BBMGCD algorithms. Both algorithms work for 31-bit primes.
4. I have created a Maple implementation of the SEPPGCD algorithm. It works for 63-bit primes.

Work to be done

1. I am currently working on exploring alternate approaches to the MHLBBPGCD algorithm. There are at least four different ways to use CMBBSHL to find $\text{pp}(g)$ using Hensel lifting.
2. I am working on additional optimizations for the Fast Euclidean algorithm in C.
3. I am still working on the implementation of the SEPPGCD algorithm.
4. I need to compute the arithmetic complexity of the MHLBBPGCD, BBMGCD, and SEPPGCD algorithms. This is not difficult in the black-box model.
5. I need to calculate the probabilities of the MHLBBPGCD, BBMGCD, and SEPPGCD algorithms returning either an incorrect gcd or FAIL. This is likely to be difficult because of the high number of failure cases.

References I

- Michael Ben-Or and Prasoon Tiwari, *A Deterministic Algorithm for Sparse Multivariate Polynomial Interpolation*, Proceedings of STOC '88, ACM, 1988, pp. 301–309.
- Tien Chen, *Sparse Hensel Lifting Algorithms for Multivariate Polynomial Factorization*, Ph.D. thesis, Simon Fraser University, 2024.
- Joachim von zur Gathen and Gerhard Jürgen, *Modern computer algebra*, 3 ed., Cambridge University Press, 2013.
- Qiao-Long Huang and Michael Monagan, *A New Sparse Polynomial GCD Algorithm by Separating Terms*, Submitted to ISSAC '24 (2024).
- Erich Kaltofen and Zhengfeng Yang, *On Exact and Approximate Interpolation of Sparse Rational Functions*, Proceedings of ISSAC '07, ACM, 2007, p. 203–210.

References II


Michael Monagan and Garrett Paluck, *New Bivariate Hensel Lifting Algorithm for n Factors*, ACM Commun. Comput. Algebra **53** (2019), no. 3, 142–145.


_____, *Linear Hensel Lifting for $\mathbb{Z}_p[x, y]$ for n Factors with Cubic Cost*, Proceedings of ISSAC '22, ACM, 2022, p. 159–166.