A New Black Box GCD Algorithm Using Hensel Lifting

Michael Monagan
Department of Mathematics
Simon Fraser University
Burnaby, BC, V5A 1S6, Canada
mmonagan@cecm.sfu.ca

Garrett Paluck
Department of Mathematics
Simon Fraser University
Burnaby, BC, V5A 1S6, Canada
gpaluck@sfu.ca

ABSTRACT

We present a new black box GCD algorithm for two multivariate polynomials a and b in $\mathbb{Z}[x_1, x_2, \ldots, x_n]$ where a and b are input as black boxes for their evaluation. Our algorithm computes $g = \gcd(a, b)$ in the sparse representation using sparse Hensel lifting from bivariate images of g. More precisely, our algorithm first computes the square-free factorization of the primitive part of g in x_1 and then, optionally, computes the content of g in x_1 recursively.

We have implemented our new algorithm in Maple with parts of it coded in C for increased efficiency. For comparison, we have implemented the Kaltofen-Diaz black box GCD algorithm and also a black box GCD algorithm constructed from the Kaltofen-Yang sparse rational function interpolation algorithm. Our experimental results show that our new algorithm is always competitive with the Kaltofen-Yang and Kaltofen-Diaz algorithms and faster when the square-free factors of g are smaller than g or we do not need the content of g in x_1.

CCS CONCEPTS

• **Computing methodologies → Symbolic and algebraic manipulation**.

ACM Reference Format:
Michael Monagan and Garrett Paluck. 2025. A New Black Box GCD Algorithm Using Hensel Lifting. In *Proceedings of the 2025 International Symposium on Symbolic and Algebraic Computation (ISSAC '25), July 28 – August 1, 2025, Guanajuato, Mexico.* ACM, New York, NY, USA, 9 pages. https://doi.org/x.x.x

1 INTRODUCTION

Let a and b be polynomials in $\mathbb{Z}[x_1, ..., x_n]$. Computing $g = \gcd(a, b)$, the greatest common divisor (GCD) of a and b, is a key operation in a Computer Algebra system. It is used to simplify the fraction a/b. The first main step to factor a is to compute $\gcd(a, \partial a/\partial x_1)$ to identify repeated factors of a.

Computing GCDs in $\mathbb{Z}[x_1, \ldots, x_n]$ is more difficult than multiplying and dividing polynomials. All variations of the Euclidean algorithm, including the Subresultant algorithm (see Brown and Traub [2]), encounter an n dimensional expression swell where the intermediate remainders grow in size. This render's those algorithms useless when n is not small. The first algorithm to avoid the expression swell was Brown's dense modular GCD algorithm from [3]. Two early GCD algorithms for sparse polynomials include Zippel's sparse modular GCD algorithm from [20] which is currently used in Fermat, Magma, Maple and Mathematica, and Wang's EEZ-GCD algorithm from [19]. Two recent sparse GCD algorithms include Hu and Monagan [10] which does a Kronecker substitution on $(x_2, ..., x_n)$ and Huang and Monagan [11] which uses a randomized Kronecker substitution.

In this work, we present a new GCD algorithm for $\mathbb{Z}[x_1, x_2, \ldots, x_n]$ where the polynomials a and b are input by black boxes for their evaluation. The black box representation was first introduced into Computer Algebra by Kaltofen and Trager in 1990 [12]. The *sparse representation* of $a \in \mathbb{Z}[x_1, ..., x_n]$ consists of a list of non-zero integer coefficients c_k and monomials M_k in $x_1, ..., x_n$ such that $a = \sum_{k=1}^{t} c_k M_k$ where t is the number of terms of a. The *black box representation* of $a \in \mathbb{Z}[x_1, ..., x_n]$ is a computer program $\mathbf{B}_a$ that takes a point $\alpha \in \mathbb{Z}^n$ and outputs $a(\alpha)$. Computing $\mathbf{B}_a(\alpha)$ is called *probing the black box*. For efficiency, we will assume we can construct a *modular black box*, that is, a black box that can compute $a(\alpha) \bmod p$ for a prime p. Thus we view the black box as mapping $\mathbf{B}_a : (\mathbb{Z}^n, p) \to \mathbb{Z}_p$. Figure 1 depicts a modular black box.

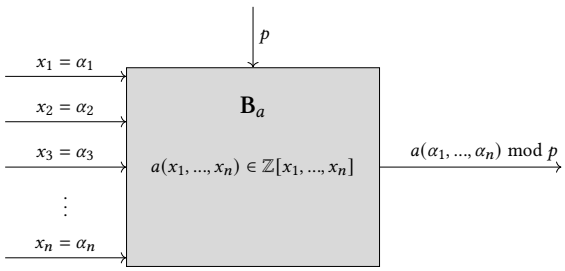

Figure 1: The modular black box model for $a \in \mathbb{Z}[x_1, ..., x_n]$

The advantage of the black box representation is that computing $\mathbf{B}_a(\alpha, p)$ can be much faster than computing $a(\alpha) \bmod p$ in the sparse representation. For example, in the black box representation the polynomial $a = (x_1 - x_2)(x_1 - x_3) \ldots (x_1 - x_n)$ can be represented using $O(n)$ space and it can be evaluated using just $n-1$ subtractions and $n-2$ multiplications. But, in the sparse representation, a has 2^{n-1} terms, which is exponential in n.

In [12], Kaltofen and Trager presented algorithms for factoring polynomials given by black boxes and computing GCDs of polynomials given by black boxes. In [7], Kaltofen and Diaz improved the black box GCD algorithm of Kaltofen and Trager [12]. The only other black box GCD algorithm that we know of is Lecerf and

ISSAC '25, July 28 – August 1, 2025, Guanajuato, Mexico

ACM ISBN x.x.x
https://doi.org/x.x.x

van der Hoeven's algorithm in [18] which uses ideas similar to the sparse rational function interpolation of Cuyt and Lee [6].

In [12], Kaltofen and Trager also presented an algorithm that, given a rational function $f = a/b$, outputs black boxes for polynomials $c = a/g/u$ and $d = b/g/u$ for some unit u. Kaltofen and Trager called this operation the separation of numerators from their denominators. In [13], Kaltofen and Yang improved on Kaltofen and Trager's algorithm. One way to compute $g = \gcd(a, b)$ is to first compute c and d then use $g = a/c$ or $g = b/d$ to obtain g. We will compare this approach using Kaltofen-Yang with our new algorithm.

Our new algorithm makes use of the Hensel lifting algorithm from Chen and Monagan [5]. It first computes $\text{pp}(g, x_1)$, the primitive part of $g = \gcd(a, b)$ in x_1, using Hensel lifting. It recovers the variables x_j for $j = 2, 3, \ldots, n$ in $\text{pp}(g, x_1)$ from bivariate images of $\text{pp}(g, x_1)$ in x_1 and x_j which are obtained by probing the black boxes for a and b. Bivariate images are used to recover the leading coefficient of $\text{pp}(g, x_1)$. Then, in a second step, it computes $\text{cont}(g, x_1)$, the content of g in x_1 recursively. An advantage of our approach is that we can easily omit computation $\text{cont}(g, x_1)$. Thus our algorithm will be faster than Kaltofen-Diaz when $\text{pp}(g, x_1)$ is smaller than g. One application where the content is not needed is when we want to solve $\{a(x_1) = 0, b(x_1) = 0\}$ for x_1.

To improve efficiency further, we recover the square-free factorization of $\text{pp}(g, x_1)$. We can do this without increasing the overall asymptotic cost by computing the square-free part of the bivariate images of g in x_1 and x_j then using our bivariate Hensel lifting from [16] to recover a square-free factorization in x_1 and x_j. This gives our algorithm a second advantage; it will be faster whenever the square-free factors of $\text{pp}(g, x_1)$ are smaller than $\text{pp}(g, x_1)$.

We have implemented our algorithm in Maple with parts implemented in C for efficiency. We use our algorithm to compute a GCD in $\mathbb{Z}[x_1, ..., x_n]$ by computing it modulo several primes and using Chinese remaindering and rational number reconstruction to recover the integer coefficients of monic(g). Our benchmarks in Section 5 show that our algorithm is faster than modified implementations of Kaltofen-Yang's and Kaltofen-Diaz's black box GCD algorithms.

2 DEFINITIONS AND NOTATION

We fix the lexicographical monomial ordering of polynomials in this paper with $x_1 > \cdots > x_n$. For a polynomial $a \in \mathbb{Z}[x_1, ..., x_n]$, we denote $\text{LC}(a)$ as the leading coefficient of a and $\text{LM}(a)$ as the leading monomial. We say a is monic if $\text{LC}(a) = 1$ and define $\text{monic}(a) = a/\text{LC}(a)$. We denote the number of terms in the polynomial a by $\#a$.

Definition 2.1. Let $a, b \in \mathbb{Z}[x_1, ..., x_n]$. Then $g = \gcd(a, b)$ is the greatest common divisor (GCD) of a and b if (i) g divides a and b, (ii) every common divisor of a and b divides g, and (iii) $\text{sign}(\text{LC}(g)) = +1$. Note (iii) imposes uniqueness on g.

Definition 2.2. Let $a \in \mathbb{Z}[x_2, ..., x_n][x_1]$ with degree $d = \deg(a, x_1)$. If $a = \sum_{i=0}^{d} a_i x_1^i$, we define $\text{LC}(a, x_1) = a_d$, a polynomial in $\mathbb{Z}[x_2, ..., x_n]$. The *content* of a in x_1 by $\text{cont}(a, x_1) = \gcd(a_0, a_1, ..., a_d)$, a polynomial in $\mathbb{Z}[x_2, ..., x_n]$. If $\text{cont}(a, x_1) = 1$, we say a is primitive in x_1. We define the *primitive part* of a in x_1 by $\text{pp}(a, x_1) = a/\text{cont}(a, x_1)$.

Definition 2.3 (Definition 8.1 in [9])**.** Let $a \in \mathbb{Z}_p[x_1, ..., x_n]$ be primitive in x_1. We say a is *square-free* if it has no repeated factors, that is, there exists no b with $\deg(b) \geq 1$ such that $b^2 | a$. The *square-free factorization* of a is $a = \prod_{i=1}^{r} a_i^i$ where each a_i is square-free and $\gcd(a_i, a_j) = 1$ for $i \neq j$. The *square-free part* of a, denoted $\text{sqf}(a)$, is defined as $\text{sqf}(a) = \prod_{i=1}^{r} a_i$.

Example 2.1. Given $g = 3(x_2 - x_3)(x_1 - x_2)^2(x_1 + x_3)$ we have $\text{cont}(g, x_1) = 3(x_2 - x_3)$, $\text{pp}(g, x_1) = (x_1 - x_2)^2(x_1 + x_3)$. The square-free factorization of $\text{pp}(g, x_1)$ is $(x_1 + x_3)^1(x_1 - x_2)^2$ and $\text{sqf}(\text{pp}(g, x_1)) = (x_1 - x_2)(x_1 + x_3)$

Our black box GCD algorithm will first compute the square-free factorization $(x_1 - x_2)^2(x_1 + x_3)$ of $\text{pp}(g, x_1)$ in factored form. It will then compute $\text{cont}(a, x_1)/3 = (x_2 - x_3)$. Here all factors have only two terms. This is faster than computing $\text{monic}(g) = (x_2 - x_3)(x_1 - x_2)^2(x_1 - x_3)$ in expanded form which has 10 terms.

Lemma 2.1. Let $a \in \mathbb{Z}[x_1, ..., x_n]$, $a \neq 0$ and $g = \gcd(a, \partial a/\partial x_1)$. Then $a/g = \text{sqf}(\text{pp}(a, x_1))$ since $\text{cont}(a, x_1) | g$.

3 OUR NEW BLACK BOX GCD ALGORITHM

Let a and b be polynomials in $\mathbb{Z}[x_1, ..., x_n]$ with associated modular black boxes $\mathbf{B}_a$ and $\mathbf{B}_b$ and let $g = \gcd(a, b)$. In this section, we present two main algorithms. Algorithm 1: MHLBBPGCD computes $\text{monic}(g)$ mod p in $\mathbb{Z}_p[x_1, ..., x_n]$ for a prime p using multivariate Hensel lifting. Algorithm 4: BBMGCD uses Chinese remaindering and rational number reconstruction (see [8, 14]) to compute $\text{monic}(g) \in \mathbb{Q}[x_1, ..., x_n]$ from modular images.

3.1 The MHLBBPGCD Algorithm

Consider the following square-free factorization

$$g = \gcd(a, b) = h \prod_{\rho=1}^{r} f_\rho^\rho$$

where (i) $h = \text{cont}(g, x_1)$ is in $\mathbb{Z}[x_2, ..., x_n]$, (ii) $\deg(f_\rho, x_1) \geq 0$, (iii) f_ρ is primitive and square-free in $\mathbb{Z}[x_2, ..., x_n][x_1]$, and (iv) $\gcd(f_i, f_j) = 1$ for $i \neq j$. We shall now describe our new algorithm which computes $\text{monic}(f_\rho)$ modulo a prime p for $1 \leq \rho \leq r$.

We assume that p is a large prime (e.g. $p = 2^{61} + 15$) chosen randomly in advance and that we have degree estimates da_i, db_i and dg_i for $\deg(a, x_i)$, $\deg(b, x_i)$, and $\deg(g, x_i)$ for $1 \leq i \leq n$. We say how we compute these estimates in Section 3.2.

We first pick an evaluation point $\boldsymbol{\alpha} = (\alpha_2, ..., \alpha_n) \in \mathbb{Z}_p^{n-1}$ at random, then interpolate $a_1 = a(x_1, \boldsymbol{\alpha})$ mod p and $b_1 = b(x_1, \boldsymbol{\alpha})$ mod p in $\mathbb{Z}_p[x_1]$ using dense interpolation and probes to the black boxes $\mathbf{B}_a$ and $\mathbf{B}_b$ and test if $\deg(a_1, x_1) = da_1$ and $\deg(b_1, x_1) = db_1$. We use at least $da_1 + 2$ probes when interpolating a_1 to check that our degree estimate da_1 is correct. We do likewise for b_1. In fact, whenever we do an interpolation in this section, we use one extra probe than necessary to check our degree estimates. If they do not agree we stop the algorithm and output FAIL.

Next we compute $g_1 = \gcd(a_1, b_1)$ in $\mathbb{Z}_p[x_1]$, check that $\deg(g_1) = dg_1$ and compute the square-free factorization of g_1. Let $g_1 = \prod_{\rho=1}^{r} \hat{f}_\rho^\rho$ be the square-free factorization of g_1 where $\hat{f}_\rho = (1/\lambda_\rho) f_\rho(x_1, \boldsymbol{\alpha})$ mod p and $\lambda_\rho = \text{LC}(f_\rho(x_1, \boldsymbol{\alpha})) \in \mathbb{Z}$ for $1 \leq \rho \leq r$. Let $\hat{h} = \lambda_h h(\boldsymbol{\alpha})$

mod p with $\lambda_h = \prod_{\rho=1}^{r} \lambda^\rho \in \mathbb{Z}$. To be explicit,

$$\begin{aligned} g(x_1, \boldsymbol{\alpha}) &= h(\boldsymbol{\alpha}) f_1(x_1, \boldsymbol{\alpha})^1 \cdots f_r(x_1, \boldsymbol{\alpha})^r \\ &= h(\boldsymbol{\alpha}) \left(\lambda_1 \hat{f}_1\right)^1 \cdots \left(\lambda_r \hat{f}_r\right)^r \text{ w.h.p.} \\ &= \underbrace{h(\boldsymbol{\alpha}) \left(\prod_{\rho=1}^{r} \lambda_\rho^\rho\right)}_{\hat{h}} \underbrace{\hat{f}_1^1 \cdots \hat{f}_r^r}_{g_1}. \end{aligned}$$

The coefficients λ_ρ can be recovered later after Chinese remaindering and the content h can be recovered recursively. For most choices of $\boldsymbol{\alpha}$ and p, we will have (i) $\deg(f_\rho(x_1, \boldsymbol{\alpha}), x_1) = \deg(\hat{f}_\rho, x_1)$, (ii) $f_\rho = \text{monic}(f_\rho(x_1, \boldsymbol{\alpha}))$ for $1 \le \rho \le r$ and (iii) $\gcd(\hat{f}_i, \hat{f}_j) = 1$ for all $i \ne j$ which is needed for Hensel lifting.

Let $g_j = \text{monic}(\text{pp}(g(x_1, ..., x_j, \alpha_{j+1}, ..., \alpha_n), x_1)) \mod p$ and let $\hat{f}_{\rho,1} = \hat{f}_\rho$. Let $\hat{f}_{\rho,j} = \text{monic}(f_\rho(x_1, ..., x_j, \alpha_{j+1}, ..., \alpha_n))$ for $2 \le j \le n$ which we will compute sequentially. We call the CMBBSHLGCD algorithm (to be described shortly) with inputs $\mathbf{B}_a, \mathbf{B}_b, \boldsymbol{\alpha}$, p, $(\hat{f}_{\rho,1}, ..., \hat{f}_{\rho,r})$, da, db, and dg to Hensel lift $\hat{f}_{\rho,1}(x_1)$ to $\hat{f}_{\rho,2}(x_1, x_2)$, then lift $\hat{f}_{\rho,2}(x_1, x_2)$ to $\hat{f}_{\rho,3}(x_1, x_2, x_3)$, etc. After the j^{th} Hensel lifting step we've computed $\hat{f}_{\rho,j}$ s.t. $\text{sqf}(g_j) = \prod_{\rho=1}^{r} \hat{f}_{\rho,j} \mod p$ and $\text{monic}(\hat{f}_{\rho,j}(x_j = \alpha_j)) = \hat{f}_{\rho,j-1} \mod p$. When the CMBBSHLGCD algorithm terminates, it returns either $\hat{f}_{\rho,n}$ such that $\text{sqf}(g_n) = \prod_{\rho=1}^{r} \hat{f}_{\rho,n} \pmod p$ or FAIL if it gets unlucky in its choice of evaluation points. If this happens, the CMBBSHLGCD algorithm must restart with a different $\boldsymbol{\alpha}$ and a new prime p. If we do not want h, the content of g, we can stop here and return $\hat{f} = \prod_{\rho=1}^{r} \hat{f}_{\rho,n}^\rho$.

To recover $\text{monic}(\text{cont}(g, x_1))$, we construct two 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 $\boldsymbol{\gamma} \in \mathbb{Z}_p^{n-1}$, $\mathbf{B}_c(\boldsymbol{\gamma}, p)$ computes $a(\beta, \boldsymbol{\gamma}) / \hat{f}(\beta, \boldsymbol{\gamma}) \mod p$ and $\mathbf{B}_d(\boldsymbol{\gamma}, p)$ computes $b(\beta, \boldsymbol{\gamma}) / \hat{f}(\beta, \boldsymbol{\gamma}) \mod p$. The black boxes return FAIL if $\hat{f}$ evaluates to 0 in which case we need to restart with a different $\boldsymbol{\alpha}$. We use our MHLBBPGCD algorithm to get the GCD of $\mathbf{B}_c$ and $\mathbf{B}_d$ over $\mathbb{Z}_p[x_2, ..., x_n]$ recursively.

We present the MHLBBPGCD algorithm as Algorithm 1.

Example 3.1. Consider the polynomials

$$a = 6(7x_2 - 3x_3)(2x_1 + 4x_2 + 1)(x_1 - x_3)^3(x_1^2 + x_2 + x_3 + 1),$$

$$b = 4(7x_2 - 3x_3)(2x_1 + 4x_2 + 1)(x_1 - x_3)^3(x_1 + x_2^2 + x_3 + 1)$$

in $\mathbb{Z}[x_1, x_2, x_3]$ and let $\mathbf{B}_a$ and $\mathbf{B}_b$ be the modular black box representations of a and b respectively. We have $g = \gcd(a, b) = 2(7x_2 - 3x_3)(2x_1 + 4x_2 + 1)(x_1 - x_3)^3$, $\text{pp}(g, x_1) = (2x_1 + 4x_2 + 1)(x_1 - x_3)^3$, $\text{cont}(g, x_1) = 2(7x_2 - 3x_3)$, and

$$\text{monic}(g) = (x_2 - \tfrac{3}{7}x_3)(x_1 + 2x_2 + \tfrac{1}{2})(x_1 - x_3)^3.$$

We demonstrate how algorithm MHLBBPGCD computes $g_m = \text{monic}(g) \mod p$ for $p = 31$. MHLBBPGCD is given degree estimates for a, b and g that are correct with high probability. In this case, the degrees are $da = (6, 3, 5)$, $db = (5, 4, 5)$ and $dg = (4, 2, 4)$ for a, b and g respectively.

Let $\boldsymbol{\alpha} = (\alpha_2, \alpha_3) = (7, 13)$. The MHLBBPGCD algorithm begins by using dense interpolation and probes to the $\mathbf{B}_a$ and $\mathbf{B}_b$ to recover

Algorithm 1: MHLBBPGCD - Computes $\text{monic}(\gcd(a, b)) \mod p$ for black boxes

Input: Modular black boxes $\mathbf{B}_a$ and $\mathbf{B}_b$ for $a, b \in \mathbb{Z}[x_1, ..., x_n]$, $X = [x_1, ..., x_n]$, $n \in \mathbb{N}$, prime p, degree estimates $da_i \le \deg(a, x_i)$, $db_i \le \deg(b, x_i)$, and $dg_i \ge \deg(\gcd(a, b), x_i)$ $(1 \le i \le n)$.

Output: $g \in \mathbb{Z}_p[x_1, x_2, ..., x_n]$ s.t. $g = \text{monic}(\gcd(a, b)) \mod p$ or FAIL.

Pick $\boldsymbol{\alpha} = (\alpha_2, ..., \alpha_n) \in (\mathbb{Z}_p \setminus \{0\})^{n-1}$ at random.
Interpolate $a_1 = a(x_1, \boldsymbol{\alpha}) \in \mathbb{Z}_p[x_1]$ via $da_1 + 2$ probes to $\mathbf{B}_a$.
if $\deg(a_1, x_1) \ne da_1$ **then return** FAIL **end**
Interpolate $b_1 = b(x_1, \boldsymbol{\alpha}) \in \mathbb{Z}_p[x_1]$ via $db_1 + 2$ probes to $\mathbf{B}_b$.
if $\deg(b_1, x_1) \ne db_1$ **then return** FAIL **end**
$g_1 \leftarrow \gcd(a_1, b_1) \in \mathbb{Z}_p[x_1]$. //$g_1$ is monic
if $\deg(g_1, x_1) \ne dg_1$ **then return** FAIL **end**
Find the square-free factorization $\prod_{\rho=1}^{r} \hat{f}_{\rho,1}^\rho$ of g_1 in $\mathbb{Z}_p[x_1]$.
if $n = 1$ **then return** $\prod_{\rho=1}^{r} \hat{f}_{\rho,1}^\rho$ **end**
//Calculate the primitive part of gcd($a \mod p$, $b \mod p$)
$[\hat{f}_{1,n}, ..., \hat{f}_{r,n}] \leftarrow$ CMBBSHLGCD($\mathbf{B}_a, \mathbf{B}_b, n, [\hat{f}_{1,1}, ..., \hat{f}_{r,1}], \boldsymbol{\alpha}, p, da, db, dg$).
if CMBBSHLGCD returned FAIL **then return** FAIL **end**
$\hat{f} \leftarrow \prod_{\rho=1}^{r} \hat{f}_{\rho,n}^{\ \rho} \in \mathbb{Z}_p[x_1, ..., x_n]$. //$\hat{f} = \text{monic}(\text{pp}(\gcd(a, b), x_1)) \mod p$
if the content of $\gcd(a, b)$ in x_1 is not needed **return** $\hat{f}$ **end**
if $dg_i - \deg(\hat{f}, x_i) = 0$ for $2 \le i \le n$ **then return** $\hat{f}$ (there is no content) **end**
// Calculate monic(cont(g, x_1)) mod p
Pick $\beta \in \mathbb{Z}_p \setminus \{0\}$ at random. //fix x_1
Create a modular black box $\mathbf{B}_c : (\mathbb{Z}^{n-1}, p) \to \mathbb{Z}_p$ which for input $\boldsymbol{\gamma} \in \mathbb{Z}_p^{n-1}$ computes $\mathbf{B}_a((\beta, \boldsymbol{\gamma}), p) / \hat{f}(\beta, \boldsymbol{\gamma}) \mod p$ if $\hat{f}(\beta, \boldsymbol{\gamma}) \ne 0$ and FAIL otherwise.
Create a modular black box $\mathbf{B}_d : (\mathbb{Z}^{n-1}, p) \to \mathbb{Z}_p$ which for input $\boldsymbol{\gamma} \in \mathbb{Z}_p^{n-1}$ computes $\mathbf{B}_b((\beta, \boldsymbol{\gamma}), p) / \hat{f}(\beta, \boldsymbol{\gamma}) \mod p$ if $\hat{f}(\beta, \boldsymbol{\gamma}) \ne 0$ and FAIL otherwise.
for i from 2 to n **do**
  $DA_i \leftarrow da_i - \deg(\hat{f}, x_i)$.
  $DB_i \leftarrow db_i - \deg(\hat{f}, x_i)$.
  $DG_i \leftarrow dg_i - \deg(\hat{f}, x_i)$.
end
$h \leftarrow$MHLBBPGCD($\mathbf{B}_c, \mathbf{B}_d, [x_2, ..., x_n], n - 1, p, DA, DB, DG$).
if MHLBBPGCD returned FAIL **then return** FAIL **end**
Set $g = h \cdot \hat{f}$.
return g.

$a_1 = a(x_1, \alpha_2, \alpha_3) \mod p = 23x_1^6 + 10x_1^5 + 21x_1^4 + 18x_1^3 + 2x_1^2 + 29x_1 + 21$, and $b_1 = b(x_1, \alpha_2, \alpha_3) \mod p = 19x_1^5 + 3x_1^4 + 4x_1^3 + 11x_1^2 + 8x_1 + 17$. After dense interpolation, we calculate

$$g_1 = \gcd(a_1, b_1) = x_1^4 + 22x_1^3 + 19x_1^2 + 24x_1 + 27.$$

Next we compute the square-free factorization of g_1 and obtain

$$g_1 = (x_1 + 30)(x_1 + 18)^3. \tag{1}$$

Next we use the CMBBSHLGCD algorithm to lift the factors of $\text{sqf}(g_1)$, namely $x_1 + 30$ and $x_1 + 18$, to get

$$\text{sqf}(\text{pp}(g_m, x_1)) = (x_1 + 2x_2 + 16)(x_1 + 30x_3) \in \mathbb{Z}_p[x_2, x_3][x_1].$$

We include the multiplicities calculated in (1) to get

$$\text{pp}(g_m, x_1) = (x_1 + 2x_2 + 16)(x_1 + 30x_3)^3.$$

Next MHLBBPGCD makes a recursive call to calculate the GCD of the polynomial contents of a and b over $\mathbb{Z}_p$. This will return $\text{cont}(g_m, x_1) = (x_2 + 4x_3)$. MHLBBPGCD concludes by returning $g_m = (x_2 + 4x_3)(x_1 + 2x_2 + 16)(x_1 + 30x_3)^3$. We note the factors of g_m are monic in lex order with $x_1 > x_2 > x_3$.

◊

3.1.1 The CMBBSHLGCD Algorithm. The CMBBSHLGCD algorithm has the following input and output:

Input: Modular black boxes $\mathbf{B}_a, \mathbf{B}_b : (\mathbb{Z}^n, p) \to \mathbb{Z}_p, \hat{f}_{\rho,1} \in \mathbb{Z}_p[x_1] (1 \le \rho \le r), \boldsymbol{\alpha} \in \mathbb{Z}_p^{n-1}$, a prime p, degree estimates $da_i \le \deg(a, x_i)$, $db_i \le \deg(b, x_i)$, and $dg_i \ge \deg(g, x_i) (1 \le i \le n)$, $X = [x_1, ..., x_n]$, and $n \in \mathbb{N}$ s.t.

(i) $\gcd(\hat{f}_{k,1}, \hat{f}_{l,1}) = 1$ for $k \ne l$ in $\mathbb{Z}_p[x_1]$,
(ii) $\text{sqf}(g_1) = \prod_{\rho=1}^{r} \hat{f}_{\rho,1} \bmod p \in \mathbb{Z}_p[x_1]$.
(iii) $\hat{f}_{\rho,1}$ is monic in x_1 for all $1 \le \rho \le r$.

Output FAIL or $\hat{f}_{\rho,n} \in \mathbb{Z}_p[x_1, ..., x_n] (1 \le \rho \le n)$ s.t.

(i) $\text{sqf}(g_n) = \prod_{\rho=1}^{r} \hat{f}_{\rho,n} \bmod p \in \mathbb{Z}_p[x_1, ..., x_n]$,
(ii) $\text{monic}(\hat{f}_{\rho,n}(x_1, \boldsymbol{\alpha})) = \hat{f}_{\rho,1} \bmod p$ for all $1 \le \rho \le r$,
(iii) $\hat{f}_{\rho,n}$ is monic in lex $x_1 > x_2 > \cdots > x_n$ for $1 \le \rho \le r$.

We have modified the CMBBSHL algorithm created by Chen and Monagan in 2024 [4]. Our algorithm lifts the monic square-free factors $\hat{f}_{\rho,1}$ of g_1 to get the monic square-free factors of $\text{pp}(g, x_1) \bmod p$. It lifts $\hat{f}_{\rho,1}(x_1)$ to $\hat{f}_{\rho,2}(x_1, x_2)$, then lifts $\hat{f}_{\rho,2}(x_1, x_2)$ to $\hat{f}_{\rho,3}(x_1, x_2, x_3)$, etc. After the j^{th} Hensel lifting step, $\text{sqf}(g_j) = \prod_{\rho=1}^{r} \hat{f}_{\rho,j} \bmod p$ and $\text{monic}(\hat{f}_{\rho,j}(x_j = \alpha_j)) = \hat{f}_{\rho,j-1} \bmod p$. After the n^{th} step, $\text{sqf}(g_n) = \prod_{\rho=1}^{r} \hat{f}_{\rho,n} \bmod p$.

CMBBSHL assumes $f_\rho(x_1, ..., x_j, \alpha_{j+1}, ..., \alpha_n)$ and $f_\rho(x_1, ..., x_{j-1}, \alpha_j, ..., \alpha_n)$ have the same supports in $x_1, ..., x_{j-1}$ for $2 \le j \le n$. This is true with high probability if p is large and α_i is chosen at random from $\mathbb{Z}_p$ (see [5]). Our CMBBSHLGCD assumes likewise.

We present the CMBBSHLGCD algorithm as Algorithm 2 and the direct sub-algorithm CMBBSHLGCDstepj as Algorithm 3.

Algorithm 2: CMBBSHLGCD

Input: Modular black boxes $\mathbf{B}_a, \mathbf{B}_b$ for $a, b \in \mathbb{Z}[x_1, ..., x_n]$, $n \in \mathbb{Z}$, $\hat{f}_{\rho,1} \in \mathbb{Z}_p[x_1] (1 \le \rho \le r)$ s.t. conditions (i)-(iii) of the input are satisfied, $\boldsymbol{\alpha} \in \mathbb{Z}_p^{n-1}$, a prime p, degree estimates da_i, db_i and dg_i for $\deg(a, x_i)$, $\deg(b, x_i)$, and $\deg(g, x_i)$ $(1 \le i \le n)$.
Output: FAIL or $\hat{f}_{\rho,n} \in \mathbb{Z}_p[x_1, ..., x_n] (1 \le \rho \le r)$ s.t. conditions (i)-(iii) of the output are satisfied.

1 **for** j from 2 to n **do**
2 $[\hat{f}_{1,j}, ..., \hat{f}_{r,j}] \leftarrow$ CMBBSHLGCDstepj($\mathbf{B}_a, \mathbf{B}_b$, $[\hat{f}_{1,j-1}, ..., \hat{f}_{r,j-1}], \boldsymbol{\alpha}, p, da, db, dg, j$). // lift x_j
3 **if** CMBBSHLGCDstepj returned FAIL **then return** FAIL **end**
4 **end**
5 **return** $[\hat{f}_{1,n}, ..., \hat{f}_{r,n}]$

In step 22 of Algorithm 3, BivariateHenselLift performs a bivariate Hensel lifting. Chen and Monagan improved our monic bivariate Hensel lifting algorithm from [16] to treat the non-monic case (see Algorithm 14 in [4]).

Algorithm 3: CMBBSHLGCDstepj: Hensel lift x_j

Input: Modular black boxes $\mathbf{B}_a, \mathbf{B}_b$ for $a, b \in \mathbb{Z}[x_1, ..., x_n]$, $\hat{f}_{\rho,j-1} \in \mathbb{Z}_p[x_1, ..., x_{j-1}] (1 \le \rho \le r)$ s.t. $\text{monic}(\text{sqf}(g_j(x_j = \alpha_j))) = \prod_{\rho=1}^{r} \hat{f}_{\rho,j-1}$, $\boldsymbol{\alpha} \in \mathbb{Z}^{n-1}$, a prime p, degree estimates da_i, db_i, dg_i, and $j \ge 2 \in \mathbb{Z}$.
Output: $\hat{f}_{\rho,j} \in \mathbb{Z}_p[x_1, ..., x_j] (1 \le \rho \le r)$ s.t. (i) $\text{sqf}(g_j) = \prod_{\rho=1}^{r} \hat{f}_{\rho,j}$, and (ii) $\text{monic}(\hat{f}_{\rho,j}(x_j = \alpha_j)) = \hat{f}_{\rho,j-1} \bmod p$ $(1 \le \rho \le r)$ or FAIL.

1 Let $\hat{f}_{\rho,j-1} = \sum_{i=0}^{df_\rho} \sigma_{\rho,i}(x_2, ..., x_{j-1}) x_1^i$ where $df_\rho = \deg(\hat{f}_{\rho,j-1}, x_1)$ for $1 \le \rho \le r$.
2 Let $\sigma_{\rho,i} = \sum_{k=1}^{s_{\rho,i}} c_{\rho,ik} M_{\rho,ik}$ where $M_{\rho,ik}$ are monomials in $x_2, ..., x_{j-1}$ and $s_{\rho,i} = \#\sigma_{\rho,i}$.
3 Pick non-zero $\beta_2, ..., \beta_{j-1} \in \mathbb{Z}_p$ at random.
4 $\mathcal{S}_{\rho,i} \leftarrow \{m_{\rho,ik} = M_{\rho,ik}(\beta_2, ..., \beta_{j-1})$ for $1 \le k \le s_{\rho,i}\}$ for $1 \le \rho \le r, 0 \le i \le df_\rho$.
5 **if** any $|\mathcal{S}_{\rho,i}| \ne s_{\rho,i}$ **return** FAIL.
6 Let s be the maximum of $s_{\rho,i}$.
 // Compute s images of the factors in $\mathbb{Z}_p[x_1, x_j]$:
7 **for** k from 1 to s **do**
8 Let $Y_k = (x_2 = \beta_2^k, ..., x_{j-1} = \beta_{j-1}^k)$.
9 Interpolate $A_k = a(x_1, Y_k, x_j, \alpha_{j+1}, ..., \alpha_n) \in \mathbb{Z}_p[x_1, x_j]$ via probes to $\mathbf{B}_a$.
10 **if** $\deg(A_k, x_1) \ne da_1$ or $\deg(A_k, x_j) \ne da_j$ **return** FAIL.
11 Interpolate $B_k = b(x_1, Y_k, x_j, \alpha_{j+1}, ..., \alpha_n) \in \mathbb{Z}_p[x_1, x_j]$ via probes to $\mathbf{B}_b$.
12 **if** $\deg(B_k, x_1) \ne db_1$ or $\deg(B_k, x_j) \ne db_j$ **return** FAIL.
13 $G_k \leftarrow \gcd(A_k, B_k) \in \mathbb{Z}_p[x_1, x_j]$.
14 **if** $\deg(G_k, x_1) \ne dg_1$ or $\deg(G_k, x_j) \ne dg_j$ **return** FAIL.
15 $S_k \leftarrow \gcd(G_k, \partial G_k / \partial x_1) \in \mathbb{Z}_p[x_1, x_j]$.
16 $G_{sf} \leftarrow \text{quo}(G_k, S_k) \in \mathbb{Z}_p[x_1, x_j]$. // $G_{sf} = \text{sqf}(G_k) \bmod p$, up to a constant in $\mathbb{Z}_p$.
17 **if** $\deg(G_{sf}, x_1) \ne \sum_{\rho=1}^{r} df_\rho$ **return** FAIL.
18 $G_{sfm} \leftarrow \text{monic}(G_{sf}) \bmod p$. // make G_{sf} monic in x_j.
19 $F_{\rho,k} \leftarrow \hat{f}_{\rho,j-1}(x_1, Y_k) \in \mathbb{Z}_p[x_1]$ for $1 \le \rho \le r$.
20 **if** any $\deg(F_{\rho,k}) < df_\rho$ (for $1 \le \rho \le r$) **return** FAIL.
21 **if** $\gcd(F_{\rho,k}, F_{\phi,k}) \ne 1$ for any $1 \le \rho < \phi \le r$ **return** FAIL.
22 $\bar{f}_{\rho,k} \leftarrow$ BivariateHenselLift($G_{sfm}, [F_{1,k}, ..., F_{r,k}], \alpha_j, p$).
23 **end**
24 **if** $n = 2$ **return** $[\bar{f}_{1,k}, ..., \bar{f}_{r,k}]$.
25 Let $\bar{f}_{\rho,k} = \sum_{l=1}^{t_\rho} \alpha_{\rho,kl} \tilde{M}_{\rho,l}(x_1, x_j) \in \mathbb{Z}_p[x_1, x_j]$ for $1 \le k \le s$, for $1 \le \rho \le r$ where $t_\rho = \#\bar{f}_{\rho,k}$.
26 **for** ρ from 1 to r **do**
27 **for** l from 1 to t_ρ **do**
28 $i \leftarrow \deg(\tilde{M}_{\rho,l}, x_1)$.
29 Solve the linear system $\{\sum_{k=1}^{s_{\rho,i}} m_{\rho,ik}^t c_{\rho,lk} = \alpha_{\rho,tl}$ for $1 \le t \le s_{\rho,i}\}$ for $c_{\rho,lk} \in \mathbb{Z}_p$.
30 **end**
31 $\hat{f}_{\rho,j} \leftarrow \sum_{l=1}^{t_\rho} (\sum_{k=1}^{s_{\rho,i}} c_{\rho,lk} M_{\rho,ik}(x_2, ..., x_{j-1})) \tilde{M}_{\rho,l}(x_1, x_j)$.
32 **end**
33 **return** $\hat{f}_{\rho,j} (1 \le \rho \le r)$

3.2 BBMGCD Algorithm (main algorithm)

The second algorithm we present in this paper is Algorithm 4, the BBMGCD algorithm. This is the main algorithm for computing $g = \text{monic}(\gcd(a, b))$ in $\mathbb{Q}[x_1, ..., x_n]$. BBMGCD takes as input the modular black boxes $\mathbf{B}_a$ and $\mathbf{B}_b$ and the set of variables $X = [x_1, ..., x_n]$. It first computes degree estimates da_i, db_i and

dg_i for $\deg(a, x_i)$, $\deg(b, x_i)$ and $\deg(g, x_i)$ respectively using Algorithm 5 defined in Section 3.2.1. The MHLBBPGCD algorithm is then called to calculate g mod p for several primes p. Chinese remaindering and rational number reconstruction (see [8, 14]) are used to obtain the rational coefficients of monic(g).

BBMGCD has two checks to make sure we have computed the correct g. It ensures the modular images of g have the same leading monomial in lex order for each different prime p and it checks if g divides both a and b probabilistically.

Algorithm 4: The BBMGCD algorithm

Input: Modular black boxes $\mathbf{B}_a$,$\mathbf{B}_b$ for $a, b \in \mathbb{Z}[x_1, ..., x_n]$, the variables $X = [x_1, ..., x_n]$.
Output: monic$(g) \in \mathbb{Q}[x_1, ..., x_n]$ s.t. $g = \gcd(a, b)$ or FAIL.

1 $M, G \leftarrow 0, 0$.
2 Pick a random 62-bit prime q for degree approximations.
3 Pick a random point $\boldsymbol{\alpha} \in \mathbb{Z}_q^n$.
4 **for** i from 1 to n **do**
5 // Use Algorithm 5 to interpolate $a_i = a(\alpha_1, ..., \alpha_{i-1}, x_i, \alpha_{i+1}, ..., \alpha_n)$ mod q and $b_i = b(\alpha_1, ..., \alpha_{i-1}, x_i, \alpha_{i+1}, ..., \alpha_n)$ mod q.
6 $a_i \leftarrow \text{UnivInterp}(\mathbf{B}_a, \boldsymbol{\alpha}, i, q)$.
7 $b_i \leftarrow \text{UnivInterp}(\mathbf{B}_b, \boldsymbol{\alpha}, i, q)$.
8 $g_i \leftarrow \gcd(a_i, b_i) \in \mathbb{Z}_q[x_i]$.
9 $da_i, db_i, dg_i \leftarrow \deg(a_i, x_i), \deg(b_i, x_i), \deg(g_i, x_i)$.
10 **end**
11 **while** *true* **do**
12 Pick a new random 62-bit prime p.
13 $g_p \leftarrow$MHLBBPGCD$(\mathbf{B}_a, \mathbf{B}_b, X, n, p, da, db, dg) \in \mathbb{Z}_p[x_1, ..., x_n]$ using Algorithm 1.
14 **if** MHLBBPGCD returned FAIL **then return** FAIL.
15 **if** $G = 0$ **then**
16 Calculate $\hat{g} \in \mathbb{Q}[x_1, ..., x_n]$ s.t. $\hat{g} \equiv g_p \pmod p$ using rational number reconstruction and set $M, G, \hat{G} \leftarrow p, g_p, \hat{g}$.
17 **else if** $\text{LM}(G) = \text{LM}(g_p)$ **then**
18 Solve $\{g^* \equiv G \pmod M, g^* \equiv g_p \pmod p\}$ for g^* using Chinese remaindering and set $M \leftarrow M \cdot p$.
19 Calculate $\hat{g} \in \mathbb{Q}[x_1, ..., x_n]$ s.t. $\hat{g} \equiv g^* \pmod M$ using rational number reconstruction.
//Require one prime of agreement
20 **if** $\hat{g} \neq$ FAIL and $\hat{g} = \hat{G}$ **then**
//Test if $\hat{g}|a$ and $\hat{g}|b$
21 Let $\hat{h} \in \mathbb{Z}[x_1, ..., x_n]$ be the result of clearing the fractions of $\hat{g}$.
22 Pick a new 63-bit prime q and point $\boldsymbol{\gamma} \in \mathbb{Z}_q^{n-1}$ at random s.t. $\deg(\hat{h}(x_1, \boldsymbol{\gamma})) = \deg(\hat{h}, x_1)$ to avoid ÷ by 0 and bad check.
23 $a_q \leftarrow \text{UnivInterp}(\mathbf{B}_a, \boldsymbol{\gamma}, 1, q)$.
24 $b_q \leftarrow \text{UnivInterp}(\mathbf{B}_b, \boldsymbol{\gamma}, 1, q)$.
25 **if** $\hat{h}(x_1, \boldsymbol{\gamma})|a_q$ and $\hat{h}(x_1, \boldsymbol{\gamma})|b_q$ in $\mathbb{Z}_q[x_1]$ **then return** $\hat{g}$ **else return** FAIL **end**
26 **end**
27 $G, \hat{G} \leftarrow g^*, \hat{g}$.
28 **else**
29 **return** FAIL.
30 **end**
31 **end**

3.2.1 Computing the degrees of a, b and g. In lines 5-8 of Algorithm 4: BBMGCD, we calculate degree estimates for $\deg(a, x_i)$, $\deg(b, x_i)$ and $\deg(g, x_i)$ for $1 \le i \le n$. To compute $\deg(a, x_i)$, we pick $\boldsymbol{\alpha} \in \mathbb{Z}_p^n$ at random then apply Algorithm 5 from [4] to interpolate $a_i = a(\alpha_1, ..., \alpha_{i-1}, x_i, \alpha_{i+1}, ..., \alpha_n)$ mod p. We have $\deg(a_i, x_i) = \deg(a, x_i)$ with high probability (see [4, 7]). It is possible that $\deg(a_i, x_i) < \deg(a, x_i)$. This is unavoidable in the black box model.

Algorithm 5: UnivInterp: Interpolate $a(x_i)$ mod p

Input: A modular black box $\mathbf{B}_a$ for $a \in \mathbb{Z}[x_1, ..., x_n]$, $\boldsymbol{\alpha} \in \mathbb{Z}_p^n$, $i \in \mathbb{N}$, and a large prime p.
Output: A polynomial $b = a(\alpha_1, ..., \alpha_{i-1}, x_i, \alpha_{i+1}, ..., \alpha_n)$ mod p s.t. $\deg(b, x_i) = \deg(a, x_i)$ w.h.p.
Reference: Kaltofen and Diaz [7], Chen and Monagan [5].

$b \leftarrow 0$; $M \leftarrow 1$; $k \leftarrow -1$.
repeat
  $k \leftarrow k + 1$.
  Pick $\beta_k \in \mathbb{Z}_p^*$ at random s.t. $\beta_k \neq \beta_j$ for $0 \le j \le k - 1$.
  $y_k \leftarrow \mathbf{B}_a((\alpha_1, \alpha_2, ..., \alpha_{i-1}, \beta_k, \alpha_{i+1}, ..., \alpha_n), p)$.
  $v_k \leftarrow (y_k - b(\beta_k))/M(\beta_k)$.
  $b \leftarrow b + v_k \cdot M$.
  $M \leftarrow M \cdot (x_i - \beta_k)$.
until $v_k = 0$;
return b.

4 COMPLEXITY ANALYSIS

Let $g = \gcd(a, b)$. In this section, we determine the complexity of Algorithm 4: BBMGCD to compute the square-free factors $f_1, f_2, ..., f_r$ of $\text{pp}(g, x_1)$. We do not include the cost of computing the square-free factors of $\text{cont}(g, x_1)$. The complexity is given in terms of the size of the inputs a and b and the outputs $f_1, f_2, ..., f_r$. Throughout the analysis $d_i = \max(\deg(a, x_i), \deg(b, x_i)$ for $1 \le i \le n$, $D = \max(d_1, ..., d_n)$, C_a and C_b are the number of arithmetic operations in $\mathbb{Z}_p$ to evaluate $\mathbf{B}_a$ and $\mathbf{B}_b$ respectively and $\#F = \sum_{\rho=1}^{r} \#f_\rho$ is the number of terms in the square-free factors.

Algorithm CMBBSHLGCD calls algorithm CMBBSHLGCDstepj $n - 1$ times to recover $x_2, x_3, ..., x_n$ one at a time in the square-free factors of $\text{pp}(g, x_1)$. This means we lose a factor of $n-1$ in efficiency when compared with algorithms like Kaltofen-Diaz which can interpolate all variables in g simultaneously. The core of Algorithm CMBBSHLGCDstepj is steps 8 to 22 whose cost is multiplied by s. Let $df_\rho = \deg(f_\rho, x_1)$ and let $f_\rho = \sum_{i=0}^{df_\rho} \tau_{\rho,i}(x_2, ..., x_n) x_1^i$. Step 2 defines

$$s_{\rho,i} = \#\sigma_{\rho,i}(x_2, ..., x_{j-1}) = \#\tau_{\rho,i}(x_2, ..., x_{j-1}, \alpha_j, ..., \alpha_n).$$

Step 6 sets $s = \max_{\rho,i} s_{\rho,i}$. Let s_j be the value of s in Algorithm CMBBSHLGCDstepj for lifting x_j and let $s_{\max} = \max_{j=2}^{n-1} s_j$. Thus $s_{\max} \le \max_{\rho,i} \#\tau_{\rho,i}$. The ratio $\#g/s_{\max}$ represents a speedup of our algorithm over an algorithm that interpolates g. In Example 2.1, $f_1 = x_1 - x_2$, $f_2 = x_1 + x_3$, $s_{\max} = 1$ and $\#g = 10$. In Example 3.1, $f_1 = x_1 + 2x_2 + \frac{1}{2}$, $f_2 = x_1 - x_3$, $s_{\max} = 2$ and $\#g = 21$.

4.1 CMBBSHLGCD Complexity

We give the following theorem for the complexity of Algorithm 2: CMBBSHLGCD.

Theorem 4.1. Let p be a large prime and, $a, b \in \mathbb{Z}[x_1, ..., x_n]$. If Algorithm CMBBSHLGCD does not return FAIL, then the total number of arithmetic operations in $\mathbb{Z}_p$ for lifting $\hat{f}_{\rho,1}$ to $\hat{f}_{\rho,n}$ using Algorithm CMBBSHLGCDstepj is at most

$$O\left((n-1)s_{\max}\left(d_1D(d_1+D+C_a+C_b) + (n+D)\#F + nD\right)\right). \quad (2)$$

From (2), one sees that the total number of probes to the black boxes is $O((n-1)\, s_{\max} d_1 D)$.

Proof. Let $da_j = \deg(a, x_j)$, $db_j = \deg(b, x_j)$, $dg_j = \deg(g, x_j)$, and $\tilde{d}_j = \deg(\mathrm{sqf}(\gcd(a, b)), x_j)$ for $1 \le j \le n$. In step 9 of Algorithm 3, we use dense interpolation to get the bivariate image $a(x_1, Y_k, x_j, \alpha_{j+1}, ..., \alpha_n)$. This operation does $O(da_1 da_j)$ probes to $\mathbf{B}_a$ and $O(da_1^2 da_j + da_1 da_j^2)$ arithmetic operations in $\mathbb{Z}_p$. The total cost of step 9 for the s interpolations is $O(sda_1 da_j C_a) + O(s(da_1^2 da_j + da_1 da_j^2)) \subseteq O(sd_1 d_j C_b) + O(s(d_1^2 d_j + d_1 d_j^2))$ arithmetic operations in $\mathbb{Z}_p$ since $d_j \ge da_j$ for $1 \le j \le n$. Similarly, the total cost of step 11 is $O(sdb_1 db_j C_b) + O(s(db_1^2 db_j + db_1 db_j^2)) \subseteq O(sd_1 d_j C_b) + O(s(d_1^2 d_j + d_1 d_j^2))$.

For step 13, we use Brown's dense GCD algorithm [3] to compute the GCD of A_k and B_k in $\mathbb{Z}_p[x_1, x_j]$. Brown's algorithm uses evaluation and interpolation on x_j. It does $O(d_1^2 d_j + d_1 d_j^2)$ arithmetic operations in $\mathbb{Z}_p$. The total cost of step 13 is $O(s(d_1^2 d_j + d_1 d_j^2))$ arithmetic operations in $\mathbb{Z}_p$.

For step 15, we again use Brown's GCD algorithm which does $O(dg_1^2 dg_j + dg_1 dg_j^2)$ arithmetic operations in $\mathbb{Z}_p$. The division in $\mathbb{Z}_p[x_1, x_j]$ in step 16 can also be done with $O(dg_1^2 dg_j + dg_1 dg_j^2)$ arithmetic operations in $\mathbb{Z}_p$ using evaluation and interpolation on x_j. Since $dg_1 \le d_1$ and $dg_j \le d_j$ the total cost of steps 15 and 16 is $O(s(d_1^2 d_j + d_1 d_j^2))$ arithmetic operations in $\mathbb{Z}_p$.

Step 19 evaluates $\hat{f}_{\rho,j-1}(x_1, \beta_2^k, ..., \beta_{j-1}^k)$ for $1 \le \rho \le r$. If we first compute the powers of β_i^k using $\sum_{i=2}^{j-1} dg_i \le (j-2)D$ multiplications, we can evaluate the terms in the factors $\hat{f}_{\rho,j-1}$ using $(j-2)\sum_{\rho=1}^{r} \#\hat{f}_{\rho,j-1}$ multiplications. Since $j - 2 < n$ and $\#\hat{f}_{\rho,j-1} \le \#f_\rho$, the total cost of step 19 is $O(snD + sn\sum_{\rho=1}^{r} \# f_\rho))$ arithmetic operations in $\mathbb{Z}_p$.

For step 22, we use Monagan and Paluck's bivariate Hensel lifting algorithm from [15, 16] which does $O(\tilde{d_1}^2 \tilde{d_j} + \tilde{d_1}\tilde{d_j}^2)$ arithmetic operations in $\mathbb{Z}_p$. The total cost of step 22 is $O(s(\tilde{d_1}^2 \tilde{d_j} + \tilde{d_1}\tilde{d_j}^2)) \subseteq O(s(d_1^2 d_j + d_1 d_j^2))$.

Using Zippel's algorithm [21], the cost of solving the Vandermonde system in step 29 for the coefficients of any given factor $\hat{f}_{\rho,j-1}$ is $\tilde{d}_j \sum_{i=0}^{df_\rho - 1} O(s_{\rho,i}^2) \subseteq O(\tilde{d}_j s \#\hat{f}_{\rho,j-1})$ since $\sum_{i=0}^{df_\rho - 1} s_{\rho,i} < \#\hat{f}_{\rho,j-1}$. The total cost to solve for the coefficients of all factors $\hat{f}_{\rho,j}$ is $O(s\tilde{d}_j \sum_{\rho=1}^{r} \#\hat{f}_{\rho,j-1}) \subseteq O(sd_j \sum_{\rho=1}^{r} \#f_\rho)$.

Summing the costs, the total number of arithmetic operations in $\mathbb{Z}_p$ for Algorithm CMBBSHLGCDstepj to recover x_j is

$$O(s(d_1^2 d_j + d_1 d_j^2 + d_1 d_j(C_a+C_b) + (n+d_j)\sum_{\rho=1}^{r} \#f_\rho + nD)). \quad (3)$$

Since $d_j \le D$, the $s \le s_{\max}$, and $\sum_{\rho=1}^{r} \#f_\rho = \#F$, summing (3) for $j = 2, 3, ..., n$ gives (2). □

4.2 MHLBBPGCD Complexity

The following theorem gives the complexity of the Algorithm 1: MHLBBPGCD for computing $\mathrm{monic}(\mathrm{pp}(\gcd(a, b), x_1))$.

Theorem 4.2. Let p be a large prime and let $a, b \in \mathbb{Z}[x_1, ..., x_n]$ and let $g = \mathrm{monic}(\mathrm{pp}(\gcd(a, b), x_1)) \bmod p$. If algorithm MHLBBPGCD returns an answer that is not FAIL, the total number of arithmetic operations in $\mathbb{Z}_p$ in the worst case for computing g using Algorithm MHLBBPGCD is

$$O\left(n\, s_{\max}\left(D^2(D + C_a + C_b) + (n + D)\#F + nD\right)\right). \quad (4)$$

From (4) the number of probes to $\mathbf{B}_a$ and $\mathbf{B}_b$ is $O(nD^2 s_{\max})$.

Proof. Step 2 of Algorithm MHLBBGCD makes $O(da_1)$ probes to $\mathbf{B}_a$ and does $O(da_1^2)$ arithmetic operations in $\mathbb{Z}_p$ to interpolate a_1 in $\mathbb{Z}_p[x_1]$. Similarly, step 4 makes $O(db_1)$ probes to $\mathbf{B}_b$ and does $O(db_1^2)$ arithmetic operations in $\mathbb{Z}_p$ to interpolate b_1 in $\mathbb{Z}_p[x_1]$. In step 6, the Euclidean algorithm does $O(da_1 db_1)$ arithmetic operations in $\mathbb{Z}_p$ to compute g_1 and step 8 needs $O(da_1 db_1)$ arithmetic operations in $\mathbb{Z}_p$ to compute the square-free factorization of g_1 (see [8]). These costs are dominated by the cost of Algorithm CMBBSHLGCD in step 10 which does $O((n-1)s_{\max}(d_1D(d_1+D+C_a+C_b) + (n+D)\#F + nD))$ arithmetic operations in $\mathbb{Z}_p$ by Theorem 4.1. The theorem follows since $d_1 \le D$. □

4.3 BBMGCD Complexity

In this section, we state the complexity of Algorithm 4: BBMGCD.

Theorem 4.3. Let $a, b \in \mathbb{Z}[x_1, ..., x_n]$ and $g = \mathrm{pp}(\gcd(a, b), x_1)$ be the primitive part of $\gcd(a, b)$ in x_1. If Algorithm BBMGCD returns an answer that is not FAIL, the total number of arithmetic operations in $\mathbb{Z}_p$ in the worst case for computing $\mathrm{monic}(g)$ is

$$O\left(\#p\, n\, s_{\max}\left(D^2(D+C_a+C_b) + (n+D)\#F + nD\right) + \#F\,\#p^2\right) \quad (5)$$

where $\#p$ is the number of primes needed to recover the rational coefficients in $\mathrm{monic}(g)$. From (5), the total number of probes to $\mathbf{B}_a$ and $\mathbf{B}_b$ is $O(\#p\, n\, D^2 s_{\max})$.

Proof. The cost of interpolating a_i in step 6 using Algorithm 4 is $O(da_i^2)$. Therefore the total cost of step 6 is $O(nda_i^2) \subseteq O(nD^2)$. Similarly, the total cost of step 7 is $O(ndb_i^2) \subseteq O(nD^2)$. The cost of step 8 is $O(d_i^2)$, so the total cost of step 8 is $O(nd_i^2) \subseteq O(nD^2)$. The costs of steps 6, 7, and 8 are dominated by the cost of step 13.

Let $\#p$ be the number of primes needed to recover the rational coefficients of $\mathrm{monic}(g)$. Since Algorithm BBMGCD uses 62 bit primes, $\#p$ is linear in the length of the largest rational number in

				BBMGCD			KY		KD	
n	$\#g$	$\#\text{pp}(g, x_1)$	$s_{\max}$	time (pp only)	eval BB	#probes	time	#probes	time	#probes
4	2	2	1	0.064 (0.064)	0.00 (0.00)	642 (642)	0.039	408	0.042	264
6	6	4	2	0.235 (0.141)	0.04 (0.00)	3984 (2690)	0.083	1690	0.100	1090
8	24	8	3	0.792 (0.306)	0.22 (0.02)	14814 (7842)	0.410	10632	0.592	6828
10	120	16	6	2.276 (0.780)	0.97 (0.05)	47548 (24242)	2.912	71594	4.503	46142
12	720	32	10	6.082 (1.770)	3.08 (0.17)	131498 (61970)	37.039	722956	56.967	468120
14	5040	64	20	16.303 (4.514)	9.47 (0.48)	358424 (176066)	2758.584	30316188	FAIL	-
16	40320	128	35	43.817 (10.688)	28.01 (1.36)	910318 (432706)	FAIL	-	FAIL	-
18	362880	256	70	124.87 (28.860)	85.92 (3.10)	2343028 (1167122)	FAIL	-	FAIL	-
20	3628800	512	126	384.785 (70.737)	290.18 (9.28)	5740202 (2788082)	FAIL	-	FAIL	-

Table 1: Benchmark 1: Timings in CPU seconds

g. The cost of using the MHLBBPGCD algorithm in step 13 is given in Theorem 4.2. Thus, the total cost of step 13 is

$$O\left(\#p\, n\, s_{\max}\left(D^2(D+C_a+C_b) + (n+D)\#FD + nD\right)\right).$$

Since the $\#p$ primes are of bounded size, the cost of Chinese remaindering and rational number reconstruction are both $O(\#p^2)$ (see [8]). So the total cost of recovering all $\#F$ rational coefficients of the factors f_ρ in steps 18 and 19 is $O(\#F\#p^2)$. The cost of steps 21 to 25 is dominated by the cost of step 13. Adding the costs of steps 13, 18 and 19 gives (5).

□

5 BENCHMARKS

We present three timing benchmarks with each benchmark executed in Maple 2024. All timings were obtained using one core on a server with 128 gigabytes of RAM and two Intel Xeon E5-2680 processors running at 2.80GHz base and 3.60GHz turbo.

Let $a, b \in \mathbb{Z}[x_1, ..., x_n]$, $g = \gcd(a, b)$, $c = a/g$ and $d = b/g$. For comparison, we've created two additional algorithms which compute monic(g). For the first algorithm, we create a modular black box for computing $f = a/b$ and use the sparse rational function interpolation algorithm proposed by Kaltofen and Yang [13] which outputs black boxes for computing $c(\boldsymbol{\sigma})$ and $d(\boldsymbol{\sigma})$ for a given point $\boldsymbol{\sigma} \in \mathbb{Z}_p^n$. From this we can compute $g(\boldsymbol{\sigma}) = a(\boldsymbol{\sigma})/c(\boldsymbol{\sigma})$. We combined this method with Ben-Or/Tiwari sparse interpolation [1] to interpolate g mod p. We then used our BBMGCD algorithm to find monic(g) using Chinese remaindering and rational number reconstruction.

For the second algorithm, we modify the black box GCD algorithm proposed by Kaltofen and Diaz in [7] to make it into a modular algorithm. Kaltofen and Diaz's algorithm creates a black box $\mathbf{B}_g$ such that $\mathbf{B}_g(\boldsymbol{\sigma})$ computes $g(\boldsymbol{\sigma})$. Instead, we create a modular black box for computing $g(\boldsymbol{\sigma})$ mod p and combine the modular black box with Ben-Or/Tiwari sparse interpolation and our BBMGCD algorithm to compute monic(g), again using Chinese remaindering and rational number reconstruction. We shall refer to these new algorithms as the "Kaltofen-Yang" algorithm and "Kaltofen-Diaz" algorithm respectively.

Our first benchmark is taken from Kaltofen-Diaz [7]. Let

$$V_1 = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_n & x_n^2 & \cdots & x_n^n \end{bmatrix}.$$

V_1 is an $n \times n$ Vandermonde matrix in the variables $[x_1, x_2, ..., x_n]$. Let V_2 be an $n \times n$ Vandermonde matrix in $[x_1, ..., x_{n/2}, y_{n/2+1}, ..., y_n]$. We create two modular black boxes $\mathbf{B}_a$, $\mathbf{B}_b : (\mathbb{Z}^n, p) \to \mathbb{Z}_p$ such that $\mathbf{B}_a$ and $\mathbf{B}_b$ return the determinants of V_1 and V_2 respectively evaluated at a point modulo a prime p. We compute the GCD of $\mathbf{B}_a$ and $\mathbf{B}_b$, namely

$$\gcd(\det(V_1), \det(V_2)) = \prod_{i=1}^{n/2-1} \prod_{j=i+1}^{n/2} (x_i - x_j). \tag{6}$$

Table 1 shows the CPU timings (in seconds) for our new algorithm compared against the Kaltofen-Yang (KY) and Kaltofen-Diaz (KD) algorithms to compute (6). All algorithms use the two primes $p = 2^{61} + 15$ and $p = 2^{61} + 21$ to compute the GCD. Column $\#g$ is the number of terms in g, column $\#\text{pp}(g, x_1)$ is the number of terms in $\text{pp}(g, x_1)$, column $s_{\max}$ is the largest s used by Algorithm 3, and column #probes is the number of times each algorithm had to probe a black box. Column "eval BB" is the time (in seconds) used by Algorithm 3 to probe the black boxes prior to bivariate interpolation (steps 9 and 11). For our BBMGCD algorithm, the timings given in brackets are the time needed to compute $\text{pp}(\gcd(\det(V_1), \det(V_2)), x_1)$ only. This illustrates the advantage of not computing $\text{cont}(g, x_1)$.

Our algorithm is faster than both the Kaltofen-Yang and Kaltofen-Diaz algorithms when $n \geq 10$. Ben-Or/Tiwari sparse interpolation is relatively fast when interpolating polynomials with a small number of terms. This partially explains why both the Kaltofen-Yang and Kaltofen-Diaz algorithm outperform our BBMGCD algorithm when n is small. In Table 1, FAIL means the Ben-Or/Tiwari sparse interpolation needed a prime greater than 2^{63} which we have not implemented and it would be slow. When computing $\text{cont}(g, x_1)$, our algorithm spends the largest amount of time on probing the black boxes for interpolations for $n \geq 12$.

Our second benchmark is taken from Monagan and Huang [11]. We want to calculate the GCD of two polynomials with $n = 5$ variables. We create the polynomial G with s terms and polynomials C and D with t terms where each monomial is chosen randomly from the set of monomials with a total degree of at most 10 and each integer coefficient is chosen randomly from $[-99, 99]$. We create the modular black boxes $\mathbf{B}_A$ and $\mathbf{B}_B$ which evaluate the polynomials $A = GC$ and $B = GD$ at a point modulo a prime p. Since G, C, D are created randomly, we have $G = \gcd(A, B)$ and G is square-free and has no polynomial content which is a worst case for our algorithm.

Table 2 uses the same algorithms and terms as Table 1.

			BBMGCD		KY		KD	
#g	t	$s_{\max}$	time	#probes	time	#probes	time	#probes
10	1000	6	0.984	17965	0.188	2113	0.204	2016
100	500	40	2.912	70631	1.121	15981	1.459	15680
250	250	73	4.409	111019	2.788	35585	3.452	34584
500	100	104	5.632	128151	6.774	70861	8.207	68654
1000	10	156	8.172	134659	19.536	141256	22.374	137048

Table 2: Benchmark 2: Timings in CPU seconds

We see a similar result as the previous benchmark. The Kaltofen-Yang and Kaltofen-Diaz algorithms outperform our BBMGCD algorithm when recovering a GCD with a relatively few terms. Our algorithm becomes faster for GCDs with many terms.

Our third benchmark is similar to the second. We generate the polynomials $G = \prod_{i=1}^{n}(x_i+1)^m$, $C = \prod_{i=1}^{n}(x_i+2)^m$, and $D = \prod_{i=1}^{n}(x_i+3)^m$ for $n = 5$ variables. We then create two modular black boxes $\mathbf{B}_A$ and $\mathbf{B}_B$ which evaluate the polynomials $A = GC$ and $B = GD$ modulo a prime p. The timings for the third benchmark are presented in Table 3.

			BBMGCD		KY		KD	
n	#g	$s_{\max}$	time	#probes	time	#probes	time	#probes
1	32	1	0.268	1872	1.627	12346	1.288	8312
2	243	1	1.351	5548	31.074	123216	22.826	82308
3	1,024	1	2.684	11160	301.603	1125026	225.901	750264
4	3,125	1	4.858	18708	1097.504	3510292	906.960	2340524
5	7,776	1	7.460	28192	3388.021	9677706	5146.178	6452216
6	16,807	1	10.296	39612	FAIL	-	FAIL	-
7	32,768	1	14.403	52968	FAIL	-	FAIL	-
8	59,049	1	19.343	68260	FAIL	-	FAIL	-
9	100,000	1	26.660	85488	FAIL	-	FAIL	-
10	161,051	1	29.823	104652	FAIL	-	FAIL	-

Table 3: Benchmark 3: Timings in CPU seconds

As our algorithm only has to lift to sqf(G) when calling the CMBBSHLGCD algorithm, it does significantly fewer probes to $\mathbf{B}_a$ and $\mathbf{B}_b$ than the Kaltofen-Yang and Kaltofen-Diaz algorithms. Benchmark 3 is a best case for our algorithm.

6 IMPLEMENTATION NOTES

We have implemented our new black box GCD algorithm in Maple with some subroutines coded in C. Our code is available for download at http://www.cecm.sfu.ca/~mmonagan/code/BBMGCD/

For Algorithm 3 CMBBSHLGCDstepj we have implemented the bivariate interpolations in steps 9 and 11 in C.

In Algorithm 3 we use Maple's GCD algorithm to compute $\gcd(a_1, b_1)$ in $\mathbb{Z}_p[x_1, x_j]$ which is coded in C.

The bivariate Hensel lift in step 22 of Algorithm 3 is coded in C. We use our algorithm from [16].

For step 28 in Algorithm 3, we solve a Vandermonde system of dimension $s_{\rho,i}$. We coded Zippel's algorithm [21] in C. It does $O(s_{\rho,i}^2)$ arithmetic operations in $\mathbb{Z}_p$.

We have performed several optimizations to our black box implementations in Benchmark 1 in order to speed up the black box probes. Our black boxes compute $\det(V_1)$ and $\det(V_2)$ evaluated at a point $\boldsymbol{\alpha} \in \mathbb{Z}_p^{2n}$ using the formula $\det(V_1) = \prod_{i=1}^{n-1}\prod_{j=i+1}^{n}(x_i - x_j)$ and similarly for $\det(V_2)$. For Benchmark 2, we do not expand the polynomials $A = CG$ and $B = DG$ in our black boxes. Instead, we evaluate C, D and G independently, then return the products.

Two places where we need to evaluate polynomials in $\mathbb{Z}_p[x_1, ..., x_n]$ are the partial factors in step 19 of Algorithm 3 and the polynomial $\hat{f}$ in steps 16 and 17 of Algorithm 1. We perform these multivariate polynomial evaluations modulo a 62 bit prime p using a C program for efficiency. Since Maple has two representations for polynomials in $\mathbb{Z}[x_1, ..., x_n]$, namely, the old SUM-OF-PROD representation and the new POLY representation (see [17]), we must handle both representations. Even with these evaluations coded in C, often more than 50% of the time is spent in these evaluations on our benchmarks. Table 4 gives a timing breakdown for the main steps of Algorithm 3 for benchmark 1 for $n = 20$.

Operation	time(s)
Compute monomial evaluations (step 4)	1.01
Probe $\mathbf{B}_a$ and $\mathbf{B}_b$ for interpolation (steps 9,11)	290.18
Perform bivariate interpolation (steps 9,11)	34.39
Compute bivariate GCD (step 13)	42.70
Compute 2nd bivariate GCD (step 15)	3.20
Compute square-free part of GCD (step 16)	0.03
Evaluate $\hat{f}_{\rho,j-1}(x_1, Y_k)$ (step 19)	1.69
Perform BivariateHenselLift (step 22)	0.40
Solve Vandermonde systems (step 29)	5.32
Other operations	5.58
Total	384.50

Table 4: Algorithm 3 breakdown for Benchmark 1 for $n = 20$

7 CONCLUSION

In this paper, we have contributed a new algorithm for computing the multivariate GCD of sparse polynomials represented by black boxes. Our benchmarks show that our algorithm is better or comparable to the Kaltofen-Yang and Kaltofen-Diaz black box GCD algorithms. We gave a complexity analysis for our new algorithm but have yet to complete a failure probability analysis.

We designed our algorithm to interpolate the square-free factors of $g = \gcd(a, b)$ which gives it an advantage when the square-free factors are smaller than g. We could design it to instead recover the irreducible factors of g over $\mathbb{Z}$ by lifting a factorization of $g(x_1, \boldsymbol{\alpha})$ over $\mathbb{Z}$. This would be faster for benchmark 1 where the factors all have 2 terms. We chose not to do this because of the additional cost of a factorizaton in $\mathbb{Z}[x]$ and because it makes the algorithm more complicated. Computing a square-free factorization is easy and does not increase the cost.

We note that the main for loop in line 7 of Algorithm 3 CMBBSHLGCDstepj can be parallelized. To reduce the number of black box probes we are looking at interpolating x_j in $G_{sf}(x_1, x_j)$ in line 17 of Algorithm 3 from images of $G_{sf}(x_1, \beta_i)$ for $\beta_i \in \mathbb{Z}_p$.

ACKNOWLEDGMENTS

This work was supported by the National Science and Research Council of Canada (NSERC) and Maplesoft.

REFERENCES

[1] Ben-Or, M., and Tiwari, P. A Deterministic Algorithm for Sparse Multivariate Polynomial Interpolation. In *Proceedings of STOC '88* (1988), ACM, pp. 301–309.

[2] Brown, W., and Traub, J. On Euclid's Algorithm and the Theory of Subresultants. *J. ACM* ***18*** (1971), 505–514.

[3] Brown, W. S. On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors. *J. ACM* ***18***, 4 (1971), 478–504.

[4] Chen, T. *Sparse Hensel Lifting Algorithms for Multivariate Polynomial Factorization.* PhD thesis, Simon Fraser University, 2024.

[5] Chen, T., and Monagan, M. A New Black Box Factorization Algorithm - the Non-Monic Case. In *Proceedings of ISSAC '23* (2023), ACM, p. 173–181.

[6] Cuyt, A., and Lee, W.-s. Sparse interpolation of multivariate rational functions. *Theoretical Computer Science* ***412***, 16 (2011), 1445–1456.

[7] Díaz, A., and Kaltofen, E. On Computing Greatest Common Divisors with Polynomials given by Black Boxes for Their Evaluations. In *Proceedings of ISSAC '95* (1995), ACM, p. 232–239.

[8] Gathen, J. v. z., and Jürgen, G. *Modern Computer Algebra*, 3 ed. Cambridge University Press, 2013.

[9] Geddes, K. O., Czapor, S. R., and Labahn, G. *Algorithms for Computer Algebra.* Kluwer Academic, 1992.

[10] Hu, J., and Monagan, M. A fast parallel sparse polynomial gcd algorithm. In *Proceedings of ISSAC '16* (2016), ACM, pp. 271–278.

[11] Huang, Q.-L., and Monagan, M. A New Sparse Polynomial GCD Algorithm by Separating Terms. In *Proceedings of ISSAC '24* (2024), ACM, pp. 134–142.

[12] Kaltofen, E., and Trager, B. M. Computing with Polynomials Given by Black Boxes for Their Evaluations: Greatest Common Divisors, Factorization, Separation of Numerators and Denominators. *J. Symb. Comput.* ***9***, 3 (1990), 301–320.

[13] Kaltofen, E., and Yang, Z. On Exact and Approximate Interpolation of Sparse Rational Functions. In *Proceedings of ISSAC '07* (2007), ACM, p. 203–210.

[14] Monagan, M. Maximal Quotient Rational Reconstruction: An Almost Optimal Algorithm for Rational Reconstruction. In *Proceedings of ISSAC '04* (2004), ACM, p. 243–249.

[15] Monagan, M. Linear Hensel Lifting for $\mathbb{Z}_p[x, y]$ and $\mathbb{Z}[x]$ with Cubic Cost. In *Proceedings of ISSAC '19* (2019), ACM, pp. 299–306.

[16] Monagan, M., and Paluck, G. Linear Hensel Lifting for $\mathbb{Z}_p[x, y]$ for n Factors with Cubic Cost. In *Proceedings of ISSAC '22* (2022), ACM, p. 159–166.

[17] Monagan, M., and Pearce, R. The design of Maple's sum-of-products and POLY data structures for representing mathematical objects. *Commun. Comput. Algebra* ***48***, 3/4 (2014), 160–186.

[18] van der Hoeven, J., and Lecerf, G. On sparse interpolation of rational functions and gcds. *Commun. Comput. Algebra* ***55***, 1 (2021), 1–12.

[19] Wang, P. S. The EEZ-GCD Algorithm. *SIGSAM Bull.* ***14***, 2 (1980), 50–60.

[20] Zippel, R. Probabilistic Algorithms for Sparse Polynomials. In *Proceedings of ISSAC '79* (1979), Springer-Verlag, p. 216–226.

[21] Zippel, R. Interpolating Polynomials from their Values. *J. Symb. Comput.* ***9***, 3 (1990), 375–403.