Computing GCDs of Multivariate Polynomials over Algebraic Number Fields Presented with Multiple Extensions

Mahsa Ansari(✉) and Michael Monagan

Department of Mathematics, Simon Fraser University,
Burnaby, BC V5A 1S6, Canada
{mansari,mmonagan}@sfu.ca

Abstract. Let $\mathbb{Q}(\alpha_1, \cdots, \alpha_n)$ be an algebraic number field. In this paper, we present a modular gcd algorithm for computing the monic gcd, g, of two polynomials $f_1, f_2 \in \mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x_1, \ldots, x_k]$. To improve the efficiency of our algorithm, we use linear algebra to find an isomorphism between $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ and $\mathbb{Q}(\gamma)$, where γ is a primitive element of $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$. This conversion is performed modulo a prime to prevent expression swell. Next, we use a sequence of evaluation points to convert the multivariate polynomials to univariate polynomials, enabling us to employ the monic Euclidean algorithm. We currently use dense interpolation to recover $x_2, \ldots, x_k$ in the gcd. In order to reconstruct the rational coefficients in g, we apply the Chinese remaindering and the rational number reconstruction. We present an analysis of the expected time complexity of our algorithm. We have implemented our algorithm in Maple using a recursive dense representation for polynomials.

Keywords: Polynomial greatest common divisors · Modular GCD algorithms · Algebraic number fields · Primitive elements

1 Introduction

1.1 Motivation for the Algorithm

Computing the gcd of polynomials is a fundamental problem in Computer Algebra, and it arises as a subproblem in many applications. For instance, computing the gcd of two polynomials plays a prominent role in polynomial factorization [14]. While the Euclidean algorithm is one of the most important algorithms for computing the gcd of two polynomials, it has a fundamental flaw for problems arising over $R[x]$ where R is not a finite field, namely, the size of the coefficients of the remainders in the Euclidean algorithm grows significantly. Especially, the Euclidean algorithm is slow when the degree of the gcd is much smaller than the degree of the inputs. The worst case occurs when the gcd of the inputs is 1. This inefficiency has led computer algebraists to develop modular gcd algorithms. Collins [3] (for univariate polynomials) and Brown [2] (for

F. Boulier et al. (Eds.): CASC 2023, LNCS 14139, pp. 1–20, 2023.
https://doi.org/10.1007/978-3-031-41724-5_1

multivariate polynomials) developed an algorithm to compute gcds by applying homomorphic reductions and Chinese remaindering. Through homomorphic reduction, they converted the gcd problem over $\mathbb{Z}$ to a simpler domain $\mathbb{Z}_p$ where the coefficients do not grow.

Let α and $\alpha_1, \ldots, \alpha_n$ be algebraic numbers. In 1989, Langemyr and MaCallum [7] designed a modular gcd algorithm for $\mathbb{Q}(\alpha)[x]$. In 1989 Smedley [13], using a different approach, designed a modular gcd algorithm for $\mathbb{Q}(\alpha)[x_1, \ldots, x_k]$. In 1995, Encarnacion [4] used rational number reconstruction [9,18] to make Langemyr and MaCallum's algorithm for $\mathbb{Q}(\alpha)[x]$ output sensitive. In 2002, Monagan and Van Hoeij [17] generalized Encarnacion's algorithm to treat polynomials in $\mathbb{Q}(\alpha_1, \cdots, \alpha_n)[x]$ for $n \geq 1$. In 2009 Li, Moreno Maza and Schost [8] used the FFT to speed up arithmetic in $\mathbb{Q}(\alpha_1, \cdots, \alpha_n)$ modulo a prime in Monagan and Van Hoeij's algorithm. State of the art algorithms for computing primitive element representations of triangular sets in softly linear time includes the works of Poteaux and Schost [11,12]. State of the art algorithms for computing in algebraic towers in softly linear time includes the work of van der Hoeven and Lecerf [15,16].

Building upon this previous work, our modular gcd algorithm, called MGCD, computes the monic gcd of two polynomials $f_1, f_2 \in \mathbb{Q}(\alpha_1, \ldots, \alpha_n)[x_1, \ldots, x_k]$ where $n \geq 1$ and $k \geq 1$. It is the first modular gcd algorithm that speeds up the computation by mapping $\mathbb{Q}(\alpha_1, \cdots, \alpha_n)$ to $\mathbb{Q}(\gamma)$ where γ is a primitive element.

1.2 Preliminaries

First, we explain relevant details and notations. Let $\mathbb{Q}(\alpha_1, \cdots, \alpha_n)$ be our number field. We build the field L as follows. Let $L_0 = \mathbb{Q}$. For $i = 1, 2, \ldots, n$ let $L_i = L_{i-1}[z_i]/\langle M_i(z_i)\rangle$ where $M_i(z_i)$ is the monic minimal polynomial of α_i over L_{i-1}. Let $L = L_n$. The field L is a $\mathbb{Q}$-vector space of dimension $d = \prod_{i=1}^{n} d_i$ where $d_i = \deg(M_i, z_i)$. Furthermore,

$$B_L = \{\textstyle\prod_{i=1}^{n}(z_i)^{e_i} \mid 0 \leq e_i < d_i\}$$

is a basis of L. Since $L \cong \mathbb{Q}(\alpha_1, \cdots, \alpha_n)$, we can perform computation over $\mathbb{Q}(\alpha_1, \cdots, \alpha_n)$ by replacing $\alpha_1, \ldots, \alpha_n$ with variables $z_1, \ldots, z_n$, respectively, and then doing the computation over L. In our algorithm, we suppose that we are given the minimal polynomials $M_1(z_1), \ldots, M_n(z_n)$ of the algebraic numbers $\alpha_1, \ldots, \alpha_n$ so that we can construct L. If $f = \sum_{e_i \in \mathbb{Z}^k_{\geq 0}} a_{e_i} X^{e_i} \in L[x_1, \ldots, x_k]$, then $a_{e_i} = \sum_{j=1}^{d} C_{e_i j} b_j$ for $b_j \in B_L$ and $C_{e_i j} \in \mathbb{Q}$. We define the **coordinate vector** of f w.r.t B_L as the vector of dimension d, denoted by $[f]_{B_L} = [v_1, \ldots, v_d]^T$, where $v_j = \sum_{e_i \in \mathbb{Z}^k_{\geq 0}} C_{e_i j} X^{e_i}$.

Example 1. We are given the field $L = \mathbb{Q}[z_1, z_2]/\langle z_1^2 - 2, z_2^2 - 3\rangle$ with basis $B_L = \{1, z_2, z_1, z_1 z_2\}$. If $f = 2z_1 x + y + z_1 + z_1 z_2 \in L[x, y]$, then $[f]_{B_L} = [y, 0, 2x+1, 1]^T$.

Let R be a commutative ring with identity $1 \neq 0$. Let us fix a monomial ordering in $R[x_1, \ldots, x_k]$. Let $f \in R[x_1, \ldots, x_k]$ and let $\mathrm{lc}(f)$ denote the leading coefficient of f and $\mathrm{lm}(f)$ denote the leading monomial of f. If $f = 0$ we define

$monic(f) = 0$. If $f \neq 0$ and $\mathrm{lc}(f)$ is a unit in R then $monic(f) = \mathrm{lc}(f)^{-1}f$. Otherwise, $monic(f) = failed$. Let $f_1, f_2 \in R[x_1, \ldots, x_k]$ and suppose a monic $g = \gcd(f_1, f_2)$ exists. Then g is unique and there exist polynomials p and q such that $f_1 = p \cdot g$ and $f_2 = q \cdot g$. We call p and q the **cofactors** of f_1 and f_2.

Example 2. Let L be as in Example 1 and $f_1 = (z_2x+z_1y)(x+y)$ and $f_2 = (z_2x+z_1y)(x-y)$ be polynomials in $L[x, y]$. By inspection, $z_2x + z_1y$ is a $\gcd(f_1, f_2)$. In lexicographical order with $x > y$ we have $\mathrm{lc}(f_1) = z_2$, $\mathrm{lm}(f_1) = x^2$ and the monic $\gcd(f_1, f_2)$ is $x + \frac{1}{3}z_1z_2y$.

Let $L_{\mathbb{Z}} = \mathbb{Z}[z_1, \ldots, z_n]$. For any $f \in L[x]$, the **denominator** of f, denoted by $den(f)$, is the smallest positive integer such that $den(f)f \in L_{\mathbb{Z}}[x]$. In addition, the **associate** of f is defined as $\tilde{f} = den(h)h$ where $h = monic(f)$. The **semi-associate** of f, denoted by $\check{f}$, is defined as rf, where r is the smallest positive rational number for which $den(rf) = 1$.

Example 3. Let L be as in Example 1 and $f = \frac{3}{2}z_1x + z_2 \in L[x]$. Then $den(f) = 2$, $\check{f} = 3z_1x + 2z_2$, $\mathrm{monic}(f) = x + \frac{1}{3}z_1z_2$ and $\tilde{f} = 3x + z_1z_2$.

To improve computational efficiency, in a preprocessing step, our modular gcd algorithm MGCD first clears fractions by replacing the input polynomials f_1 and f_2 with their semi-associates. Computing associates can be expensive when $\mathrm{lc}(f_1)$ and $\mathrm{lc}(f_2)$ are complicated algebraic numbers. Thus, we prefer to use semi-associates instead of associates to remove fractions. Then, MGCD computes $\gcd(f_1, f_2)$ modulo a sequence of primes.

Definition 1. *Let p be a prime such that $p \nmid \prod_{i=1}^{n} \mathrm{lc}(\check{M}_i) \cdot \mathrm{lc}(\check{f}_1)$. Let $m_i(z_i) = M_i \bmod p$ for $1 \leq i \leq n$. Define $L_p = \mathbb{Z}_p[z_1, \ldots, z_n]/\langle m_1, \ldots, m_n \rangle$.*

L_p is a finite ring with p^d elements which likely has zero divisors. We give an example of MGCD to illustrate the treatment of zero-divisors in L_p and to motivate the use a primitive element.

Example 4. We continue Example 2 where $L = \mathbb{Q}[z_1, z_2]/\langle z_1^2 - 2, z_2^2 - 3 \rangle$, $f_1 = (z_2x + z_1y)(x + y)$, $f_2 = (z_2x + z_1y)(x - y)$ and $g = x + \frac{1}{3}z_1z_2y$ is the monic $\gcd(f_1, f_2)$. Suppose MGCD picks $p = 3$. Then $m_1 = z_1^2 + 1$, $m_2 = z_2^2$ and

$$L_3 = \mathbb{Z}_3[z_1, z_2]/\langle\, z_1^2 + 1, z_2^2 \,\rangle.$$

Notice that z_2 is a zero divisor in L_3. Next, MGCD picks an evaluation point $\alpha \in \mathbb{Z}_p$ and attempts to compute $\gcd(f_1(x, \alpha), f_2(x, \alpha))$ in $L_3[x]$ using the monic Euclidean algorithm (MEA) (see [17]). The MEA will try to compute $r_1 = monic(f_2(x, \alpha))$ and then divide $r_0 = f_1(x, \alpha)$ by r_1 but $monic(f_2(x, \alpha))$ fails as $\mathrm{lc}(f_2(x, \alpha)) = z_2$ is a zero-divisor in L_3. Since MGCD does know whether this is because of the choice of p or α, it stops the computation of $\gcd(f_1, f_2)$ modulo $p = 3$ and tries another prime, for example, $p = 5$. We have

$$L_5 = \mathbb{Z}_5[z_1, z_2]/\langle\, z_1^2 + 3, z_2^2 + 2 \,\rangle.$$

Once again, MGCD chooses $\alpha \in \mathbb{Z}_5$ and computes $\gcd(f_1(x,\alpha), f_2(x,\alpha))$ in $L_5[x]$ using the MEA. This time $\mathrm{lc}(f_2(x,\alpha)) = z_2$ is a unit in L_5 with inverse $2z_2$ and $monic(f_2(x,\alpha))$ succeeds. The MEA also succeeds and outputs $g_5 = x + 2\alpha z_1 z_2$. Notice $g_5 = g(x,\alpha) \mod 5$. MGCD repeats this process for more α's and primes and recovers $g = x + \frac{1}{3} z_1\, y$ using polynomial interpolation for y and Chinese remaindering and rational number reconstruction [9,18] for the fraction $\frac{1}{3}$.

Most of the computational work in MGCD occurs in the finite ring L_p. To speed up MGCD we use a primitive element to speed up arithmetic in L_p. We note that our Maple implementation of MGCD uses 31-bit primes which avoids zero divisors in L_p with high probability.

1.3 Paper Outline

Our paper is organized as follows. In Sect. 2, we use the fact that $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ can be specified as a $\mathbb{Q}$-vector space to compute a primitive element γ for $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$. We also construct a ring isomorphism ϕ_γ between the quotient rings L_p and $\bar{L}_p = \mathbb{Z}_p[z]/\langle M(z)\rangle$ where p is a prime and $M(z) \in \mathbb{Z}_p[z]$ is the minimal polynomial for $\gamma \bmod p$. In our modular gcd algorithm, we apply ϕ_γ to speed up arithmetic in L_p. In Sect. 3, we describe the PGCD algorithm for computing the monic gcd of two polynomials $f_1, f_2 \in \bar{L}_p[x_1, \ldots, x_k]$, where $k \geq 2$. We then present our modular gcd algorithm, MGCD. In Sect. 4, we study the expected time complexity of our MGCD algorithm. Finally, in Sect. 5, we present an implementation of our algorithm in Maple which uses the recursive dense polynomial data structure described in [17]. We then present a timing benchmark for running Algorithm MGCD. Our Maple code is available at http://www.cecm.sfu.ca/~mmonagan/code/MGCD.

2 Converting $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ to a Single Extension $\mathbb{Q}(\gamma)$

The main goal of this section is to identify a primitive element for $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ called γ and compute its minimal polynomial. We then proceed to reduce the computation of finding γ modulo a prime p, which allows us to form the quotient ring $\bar{L}_p = \mathbb{Z}_p[z]/\langle M(z)\rangle$ where $M(z)$ is the minimal polynomial of γ modulo p. Once we have constructed $\bar{L}_p$, we determine the ring isomorphism $\phi_\gamma : L_p \longrightarrow \bar{L}_p$. We use ϕ_γ in our MGCD algorithm to map a polynomial over the multiple extension L_p to its corresponding polynomial over the simple extension $\bar{L}_p$.

2.1 Computing a Primitive Element and its Minimal Polynomial

In order to find a primitive element for $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$, we start by choosing random integers $C_1, \ldots, C_{n-1}$ from the interval $[1, p)$, where p is a large prime. Using these integers, we create a potential primitive element $\gamma = \alpha_1 + \sum_{i=2}^{n} C_{i-1}\alpha_i$. To determine whether γ is a primitive element or not we use Theorem 1.

Theorem 1. *Let $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ have degree d and let $C_1, \ldots, C_{n-1} \in \mathbb{Z}$ be chosen randomly from $[1, p)$ where p is a large prime. Define $\gamma = \alpha_1 + \sum_{i=2}^{n} C_{i-1}\alpha_i$, and let B be a basis for $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ as a $\mathbb{Q}$-vector space. Let A be the $d \times d$ matrix whose ith column is $[\gamma^{i-1}]_B$ for $1 \leq i \leq d$. Then, γ is a primitive element for $\mathbb{Q}(\alpha_1, \cdots, \alpha_n) \Longleftrightarrow \det(A) \neq 0$.*

Proof. ($\Longrightarrow$) If γ is a primitive element for $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$, then we have $[\mathbb{Q}(\gamma) : \mathbb{Q}] = [\mathbb{Q}(\alpha_1, \ldots, \alpha_n) : \mathbb{Q}] = d$. Let $B_K = \{1, \gamma, \ldots, \gamma^{d-1}\}$ be a basis for $K = \mathbb{Q}(\gamma)$ as a $\mathbb{Q}$-vector space. Since $\mathbb{Q}(\alpha_1, \ldots, \alpha_n) = K$, any element of B_K can be expressed as a linear combination of elements of B. Thus, the $d \times d$ linear system

$$\begin{aligned} 1 &= c_{11}b_1 + c_{12}b_2 + \ldots + c_{1d}b_d \\ \gamma &= c_{21}b_1 + c_{22}b_2 + \ldots + c_{2d}b_d \\ &\ldots \\ \gamma^{d-1} &= c_{d1}b_1 + c_{d2}b_2 + \ldots + c_{dd}b_d \end{aligned}$$

has a unique solution. We can form the $d \times d$ matrix D, whose ith row is $[\gamma^{i-1}]_B^T$ for $1 \leq i \leq d$. Since the above system of equations has a unique solution, the matrix D is invertible, and thus $\det(D) \neq 0$. On the other hand, $D = A^T$ so

$$0 \neq \det(D) = \det(A^T) = \det(A).$$

($\Longleftarrow$) Given $\det(A) \neq 0$, we can conclude that A is invertible and the linear system $A \cdot q = -[\gamma^d]_B$ has a unique solution $q = [q_1, \ldots, q_d]^T$. If we prove that the polynomial of degree d

$$M(z) = z^d + \sum_{i=1}^{d} q_i z^{i-1}$$

is the minimal polynomial of γ, then $[\mathbb{Q}(\gamma) : \mathbb{Q}] = [\mathbb{Q}(\alpha_1, \ldots, \alpha_n) : \mathbb{Q}] = d$ which implies that γ is a primitive element as required. By construction, $M(z)$ is monic, $\deg(M(z)) = d$, and $M(\gamma) = 0$. Hence, we only need to prove that $M(z)$ is irreducible over $\mathbb{Q}$. Suppose that $M(z)$ is reducible. Since $\mathbb{Q}[z]$ is a UFD, $M(z)$ can be expressed as a product of monic irreducible polynomials over $\mathbb{Q}$, i.e. $M(z) = p_1(z) \cdots p_k(z)$ where each $p_i(z) \in \mathbb{Q}[z]$ is irreducible for $1 \leq i \leq n$. Since $M(\gamma) = 0$, there exists $1 \leq i \leq k$ such that $p_i(\gamma) = 0$ which implies that $p_i(z)$ is the minimal polynomial of γ. Let $\deg(p_i(z)) = h$ so $\{1, \gamma, \ldots, \gamma^{h-1}\}$ forms a basis for $\mathbb{Q}(\gamma)$. Hence, $\{1, \gamma, \ldots, \gamma^{d-1}\} \subseteq Span(\{1, \gamma, \ldots, \gamma^{h-1}\})$ where $h < d$. That is, the set $\{1, \gamma, \ldots, \gamma^{d-1}\}$ is a linearly dependant set, equivalently, the matrix A has two or more linearly dependent columns which means $\det(A) = 0$. This contradicts the assumption that $\det(A) \neq 0$. Therefore, $M(z)$ must be irreducible over $\mathbb{Q}$, and hence it is the minimal polynomial of γ.

We can employ Theorem 1 to compute the minimal polynomial of the primitive element γ.

Corollary 1. *Under the assumptions of Theorem 1, if* $\det(A) \neq 0$ *and* $q = [q_1, \ldots, q_d]^T$ *be the solution of the linear system* $A \cdot q = -[\gamma^d]_B$*, the polynomial* $M(z) = z^d + \sum_{i=1}^{d} q_i z^{i-1}$ *is the minimal polynomial of* γ*.*

Proof. Corollary 1 follows directly from the proof of Theorem 1.

We present Algorithm 1, LAminpoly, which is used to verify if $\gamma = \alpha_1 + \sum_{i=2}^{n} C_{i-1}\alpha_i$, where $C_i \in \mathbb{Z}$ for $2 \leq i \leq n$, is a primitive element for $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$. LAminpoly can be run over two different ground fields: $F = \mathbb{Q}$ and $F = \mathbb{Z}_p$, where p is a prime. If LAminpoly does not fail over $F = \mathbb{Q}$, according to Theorem 1 and Corollary 1, γ is a primitive element for $\mathbb{Q}(\alpha_1, \ldots, \alpha_n)$ and the output $M(z)$ is the minimal polynomial of γ. In the following example, we execute the LAminpoly algorithm over $F = \mathbb{Q}$.

Algorithm 1: LAminpoly

Input: A list of the minimal polynomials $[M_1(z_1), \ldots, M_n(z_n)]$, the ground field F over which the computation is performed, and $\gamma = z_1 + C_1 z_2 + \ldots + C_{n-1} z_n$ where $C_i \in \mathbb{Z}$ for $1 \leq i \leq n-1$

Output: Either a message "FAIL" or a polynomial $M(z) \in F[z]$ such that $M(\gamma) = 0$, the matrix A and A^{-1} .

$B_L = \{ \prod_{i=1}^{n} (z_i)^{e_i} \ 0 \leq e_i < d_i \}$ s.t $d_i = \deg(M_i(z_i))$ // A basis for L
$d = \prod_{i=1}^{n} d_i$
Initialize A to be a $d \times d$ zero matrix over F.
$g_0 = 1$
for $i = 1$ *to* d **do**
    Set column i of A to be $[g_{i-1}]_{B_L}$
    $g_i = \gamma \cdot g_{i-1}$
if $\det(A) = 0$ **then**
    return(FAIL)
Compute A^{-1}
Solve the $d \times d$ linear system $A \cdot q = -[g_d]_{B_L}$ for q
Construct the polynomial $M(z) := q_1 + q_2 z + \ldots + q_d z^{d-1} + z^d$
return($M(z)$, A, A^{-1})

Example 5. Let $M_1(z_1) = z_1^2 - 2$ be the minimal polynomial of $\sqrt{2}$ over $\mathbb{Q}$ and $M_2(z_2) = z_2^2 - 3$ be the minimal polynomial of $\sqrt{3}$ over $\mathbb{Q}[z_1]/\langle z_1^2 - 2\rangle$. Let $L = \mathbb{Q}[z_1, z_2]/\langle z_1^2 - 2, z_2^2 - 3\rangle$. Let $C_1 = 1$ so that $\gamma = z_1 + z_2$. We wish to test if γ is a primitive element. Let $B_L = \{1, z_2, z_1, z_1 z_2\}$ and $B_K = \{1, z, z^2, z^3\}$ be the bases for L and $K = \mathbb{Q}[z]/\langle M(z)\rangle$ respectively, where $M(z)$ is the minimal polynomial of γ. Let $a_i = [\gamma^i]_{B_L}$ be the coordinate vector of γ^i relative to B_L for $0 \leq i \leq 4$. Then we have

$$a_0, a_1, a_2, a_3, a_4 = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix}, \begin{bmatrix} 0 \\ 1 \\ 1 \\ 0 \end{bmatrix}, \begin{bmatrix} 5 \\ 0 \\ 0 \\ 2 \end{bmatrix}, \begin{bmatrix} 0 \\ 9 \\ 11 \\ 0 \end{bmatrix}, \begin{bmatrix} 49 \\ 0 \\ 0 \\ 20 \end{bmatrix}.$$

The coefficient matrix A is the 4×4 matrix containing a_0, a_1, a_2, a_3 as its columns

$$A = \begin{bmatrix} 1 & 0 & 5 & 0 \\ 0 & 1 & 0 & 9 \\ 0 & 1 & 0 & 11 \\ 0 & 0 & 2 & 0 \end{bmatrix}, \; A^{-1} = \begin{bmatrix} 1 & 0 & 0 & -\frac{5}{2} \\ 0 & \frac{11}{2} & -\frac{9}{2} & 0 \\ 0 & 0 & 0 & \frac{1}{2} \\ 0 & -\frac{1}{2} & \frac{1}{2} & 0 \end{bmatrix}.$$

As $\det(A) = -4$, we conclude that $C_1 = 1$ is an appropriate constant and $\gamma = z_1 + z_2$ is a primitive element. The next step is to compute $M(z)$. Applying Corollary 1 we have $q = A^{-1}(-a_4) = [1, 0, -10, 0]^T$ thus $M(z) = z^4 - 10z^2 + 1$.

If we execute Algorithm 1 over $F = \mathbb{Z}_p$, then we can use the resulting polynomial $M(z)$ and matrix A to construct $\bar{L}_p = \mathbb{Z}_p[z]/\langle M(z)\rangle$ such that $L_p \cong \bar{L}_p$. However, if we execute LAminpoly over $F = \mathbb{Z}_p$, it is likely that one or more of m_i will be reducible over $\mathbb{Z}_p[z_1, \ldots, z_{i-1}]/\langle m_1, \ldots, m_{i-1}\rangle$ in which case $M(z)$ is reducible over $\mathbb{Z}_p$. We give an example.

Example 6. Let $M_1(z_1) = z_1^2 - 2$ and $M_2(z_2) = z_2^2 - 3$ and $L = \mathbb{Q}[z_1, z_2]/\langle M_1, M_2\rangle$. Let $p = 113$, $F = \mathbb{Z}_p$, $C_1 = 101$ and $L_p = \mathbb{Z}_{113}[z_1, z_2]/\langle z_1^2 + 111, z_2^2 + 110\rangle$. L_p is not a field since $m_1 = (z_1 + 51)(z_1 + 62)$ in L_p. Let $B_{L_p} = \{1, z_2, z_1, z_1z_2\}$. Applying LAminpoly for $\gamma = z_1 + 101z_2 \in L_p$ we have

$$A = \begin{bmatrix} 1 & 0 & 95 & 0 \\ 0 & 101 & 0 & 55 \\ 0 & 1 & 0 & 55 \\ 0 & 0 & 89 & 0 \end{bmatrix}.$$

Since $\det(A) \neq 0$, we solve the system $A\,q = -[\gamma^4]_{B_{L_p}}$ and construct the generator polynomial $M(z) = z^4 + 36z^2 + 32$. $M(z)$ factors over $\mathbb{Z}_p$ as $M(z) = \left(z^2 + 11z + 22\right)\left(z^2 + 102z + 22\right)$ so $\bar{L}_p = \mathbb{Z}_p[z]/\langle M(z)\rangle$ is not a field.

Remark 1. In MGCD, we choose a prime p and $C_1, \ldots, C_{n-1} \in [1, p)$ at random. Then we call algorithm LAminpoly with $F = \mathbb{Z}_p$ and $\gamma = z_1 + C_1z_2 + \ldots C_{n-1}z_n$ in L_p. If LAminpoly returns FAIL, because the failure may be due to the choice of p or $C_1, \ldots, C_{n-1}$, MGCD selects a new prime p and a new set of random integers $C_1, \ldots, C_{n-1} \in [1, p)$ and calls LAminpoly again.

2.2 The Isomorphism ϕ_γ

We are now well-equipped to introduce the isomorphism $\phi_\gamma : L_p \longrightarrow \bar{L}_p$.
Let $B_{L_p} = \{\prod_{i=1}^{n}(z_i)^{e_i} \; s.t \; 0 \leq e_i < d_i\}$ and $B_{\bar{L}_p} = \{1, z, z^2, \ldots, z^{d-1}\}$ be bases for L_p and $\bar{L}_p$, respectively. Let $C : L_p \longrightarrow \mathbb{Z}_p^d$ be a bijection such that $C(a) = [a]_{B_{L_p}}$ and $D : \bar{L}_p \longrightarrow \mathbb{Z}_p^d$ be another bijection such that $D(b) = [b]_{B_{\bar{L}_p}}$. Define $\phi_\gamma : L_p \longrightarrow \bar{L}_p$ such that $\phi_\gamma(a) = D^{-1}(A^{-1} \cdot C(a))$, where A is the matrix obtained from the LAminpoly algorithm over $F = \mathbb{Z}_p$. The inverse of ϕ_γ is $\phi_\gamma^{-1} : \bar{L}_p \longrightarrow L_p$ such that $\phi_\gamma^{-1}(b) = C^{-1}(A \cdot D(b))$.

Lemma 1. *If* $\det(A) \neq 0$*, then the mapping* ϕ_γ *defined above is a ring isomorphism.*

Proof. Since A^{-1} exists and both C and D are bijections, we can conclude that ϕ_γ is well-defined and bijective. Additionally, if $\gamma = z_1 + C_1 z_2 + \cdots + C_{n-1} z_n$ is the element obtained from the LAminpoly algorithm, then ϕ_γ^{-1} can be expressed as an evaluation homomorphism that substitutes z for $z_1 + C_1 z_2 + \ldots + C_{n-1} z_n$. The fact that ϕ_γ^{-1} is a homomorphism implies that ϕ_γ is also a ring homomorphism.

Isomorphism ϕ_γ induces the natural isomorphism $\phi_\gamma : L_p[x_1, \ldots, x_k] \longrightarrow \bar{L}_p[x_1, \ldots, x_k]$. The following example illustrates how we can compute $\phi_\gamma(f)$ for $f \in L_p[x_1, \ldots, x_k]$.

Example 7. Given the quotient rings $L_p = \mathbb{Z}_{113}[z_1, z_2]/\langle z_1^2 + 111, z_2^2 + 110\rangle$ and $\bar{L}_p = \mathbb{Z}_{113}[z]/\langle z^4 + 36z^2 + 32\rangle$ from Example 6, we aim to compute $\phi_\gamma(f)$ where $f = 2x_1z_1 + x_2 + z_1z_2 \in L_p[x_1, x_2]$. Let $B_{L_p} = \{1, z_2, z_1, z_1z_2\}$ and A be the matrix computed in Example 6. We have $[f]_{L_p} = [x_2, 0, 2x_1, 1]^T$ and

$$b = A^{-1} \cdot [f]_{B_{L_p}} = [x_2 + 84, 61x_1, 80, 77x_1]^T$$

as the coordinate vector of $\phi_\gamma(f)$ relative to $B_{\bar{L}_p} = \{1, z, z^2, z^3\}$. Consequently,

$$\phi_\gamma(f) = x_2 + 84 + 61x_1z + 80z^2 + 77x_1z^3 \in \bar{L}_p[x_1, x_2].$$

3 The Modular Gcd Algorithm

Modular gcd algorithms for $\mathbb{Q}(\alpha_1, \cdots, \alpha_n)[x]$ work by computing the gcd modulo a sequence of primes and applying Chinese remaindering and rational number reconstruction to recover the rational coefficients of the gcd. However, not all primes can be used. Our modular gcd algorithm for $\mathbb{Q}(\alpha_1, \cdots, \alpha_n)[x_1, \ldots, x_n]$ applies Theorem 2 below to identify the primes that cannot be used. In Theorem 2, R' may have zero-divisors. Examples 8 and 9 illustrate this.

Theorem 2. *Let* R *and* R' *be commutative rings with* $1 \neq 0$ *and* $\phi : R \longrightarrow R'$ *be a ring homomorphism. Let* f_1 *and* f_2 *be two non-zero polynomials in* $R[x_1, \ldots, x_k]$*. Let us fix a monomial ordering on* $R[x_1, \ldots, x_k]$*. Suppose that the monic* $g = \gcd(f_1, f_2)$ *and the monic* $g_\phi = \gcd(\phi(f_1), \phi(f_2))$ *exist. If* $\phi(\mathrm{lc}(f_1)) \neq 0$*, then*

(i) $\mathrm{lm}(g_\phi) \geq \mathrm{lm}(g)$ *and*
(ii) If $\mathrm{lm}(g_\phi) = \mathrm{lm}(g)$*, then* $g_\phi = \phi(g)$*.*

Proof. (i) Let $p, q \in R[x_1, \ldots, x_k]$ be the cofactors of f_1 and f_2, respectively. That is, $f_1 = p \cdot g$ and $f_2 = q \cdot g$. Using the ring homomorphism property of ϕ, we have $\phi(f_1) = \phi(p) \cdot \phi(g)$ and $\phi(f_2) = \phi(q) \cdot \phi(g)$. By assumption, $\phi(\mathrm{lc}(f_1)) \neq 0$ which implies that $\phi(f_1) \neq 0$. Furthermore, since $\phi(\mathrm{lc}(g)) = \phi(1) = 1$, we have $\phi(g) \neq 0$. Thus, $\phi(g)$ is a common factor of

$\phi(f_1)$ and $\phi(f_2)$, and hence $\phi(g) \mid g_\phi$. In other words, there exists a non-zero polynomial $h \in R'[x_1, \ldots, x_k]$ such that $g_\phi = h \cdot \phi(g)$. If $\mathrm{lc}(h) \cdot \mathrm{lc}(\phi(g)) = 0$, then $\mathrm{lc}(h) \cdot 1 = 0$, which implies that $\mathrm{lc}(h) = 0$, contradicting the assumption that $h \neq 0$. Accordingly, $\mathrm{lm}(g_\phi) = \mathrm{lm}(h) \cdot \mathrm{lm}(\phi(g))$ which implies that $\mathrm{lm}(g_\phi) \geq \mathrm{lm}(\phi(g))$. Moreover, since $\phi(\mathrm{lc}(g)) = \phi(1) = 1$, we have $\mathrm{lm}(\phi(g)) = \mathrm{lm}(g)$ and hence $\mathrm{lm}(g_\phi) \geq \mathrm{lm}(\phi(g)) = \mathrm{lm}(g)$.

(ii) To prove the second part, we use the fact that $g_\phi = h \cdot \phi(g)$ and the assumption that $\mathrm{lm}(g_\phi) = \mathrm{lm}(g)$ to conclude that $\mathrm{lm}(h) = 1$. Thus, h is a constant and since both $\phi(g)$ and g_ϕ are monic, $h = 1$. Hence, $g_\phi = \phi(g)$.

3.1 PGCD

Algorithm **PGCD** (see Algorithm 2) computes the monic $\gcd(f_1, f_2)$, where $f_1, f_2 \in \bar{L}_p[x_1, \ldots, x_k]$ for $k \geq 1$. We use evaluation and dense interpolation as in [2]. PGCD is recursive. When $k = 1$ we employ the monic Euclidean algorithm [17] to find $\gcd(f_1, f_2) \in \bar{L}_p[x_1]$. Otherwise, PGCD reduces f_1, f_2 to polynomials in $\bar{L}_p[x_1, \ldots, x_{k-1}]$ by evaluating $x_k = b_k$ where b_k is chosen randomly from $\mathbb{Z}_p$. Then, PGCD computes

$$\gcd(f_1(x_1, x_2, \ldots, x_{k-1}, b_k), f_2(x_1, x_2, \ldots, x_{k-1}, b_k))$$

recursively. Subsequently, PGCD interpolates x_k in g. It interpolates x_k incrementally until the interpolated polynomial H does not change. The condition in line 30 implies this.

Let $R = \bar{L}_p[x_k]$ and $R' = \bar{L}_p$. We define the evaluation homomorphism $\phi_{x_k=b} : R[x_1, \ldots, x_{k-1}] \longrightarrow R'[x_1, \ldots, x_{k-1}]$ such that $\phi_{x_k=b}(f) = f(b)$. The chosen evaluation points may cause several problems, including the possibility of hitting a zero divisor. Here, we identify four types of evaluation points.

Definition 2. *We consider f_1 and f_2 as polynomials in $\bar{L}_p[x_k][x_1, \ldots, x_{k-1}]$ so that $\mathrm{lc}(f_1) \in \bar{L}_p[x_k]$ and $\mathrm{lm}(f_1)$ is a monomial in $x_1, \ldots, x_{k-1}$. Assume that the monic $g = \gcd(f_1, f_2)$ exists. Let $b \in \mathbb{Z}_p$ be an evaluation point. We distinguish the following cases:*

- ***Lc-bad Evaluation Points**. We call b an lc-bad evaluation point if $\mathrm{lc}(f_1)(b)=0$.*
- ***Zero-Divisor Evaluation Points**. If b is not an lc-bad evaluation point, and the monic Euclidean algorithm (see [17]) tries to invert a zero-divisor in $\bar{L}_p$, for the evaluated f_1 and f_2 at $x_k = b$, then b is called a zero-divisor evaluation point.*
- ***Unlucky Evaluation Points**. Assume the monic $\gcd(\phi_{x_k=b}(f_1), \phi_{x_k=b}(f_2))$, denoted by g_b, exists. We call b an unlucky evaluation point if $\mathrm{lm}(g_b) > \mathrm{lm}(g)$.*
- ***Good Evaluation Points**. If b is neither lc-bad, unlucky, nor zero-divisor evaluation point, we call b a good evaluation point.*

Theorem 3. *Let* $\phi_{x_k=b} : R[x_1, \ldots, x_{k-1}] \longrightarrow R'[x_1, \ldots, x_{k-1}]$ *be the evaluation homomorphism, where* $R = \bar{L}_p[x_k]$ *and* $R' = \bar{L}_p$. *Let* $f_1, f_2 \in R[x_1, \ldots, x_{k-1}]$ *and* $b \in \mathbb{Z}_p$. *Suppose that*

$$\begin{aligned} g &= monic(\gcd(f_1, f_2)) \\ g_b &= monic(\gcd(\phi_{x_k=b}(f_1), \phi_{x_k=b}(f_2))) \\ h &= monic(\phi_{x_k=b}(g)) \end{aligned}$$

all exist. If b *is a good evaluation point, then* $h = g_b$.

Proof. If b is a good evaluation point, then it is not lc-bad. Thus, we can infer that $\phi_{x_k=b}(\mathrm{lc}(f_1)) \neq 0$. By a similar argument as in the proof of Theorem 2, we can conclude that h is a common factor of $\phi_{x_k=b}(f_1)$ and $\phi_{x_k=b}(f_2)$ so $h \mid g_b$. In other words, there is a non-zero polynomial $t \in R'[x_1, \ldots, x_{k-1}]$ such that $g_b = t \cdot h$. Since h is monic, the same justification in Theorem 2 leads us to conclude that $\mathrm{lm}(g_b) \geq \mathrm{lm}(h)$. On the other hand, by the definition of a good evaluation point, b is not an unlucky evaluation point. Thus, we can conclude that $\mathrm{lm}(g_b) = \mathrm{lm}(h)$. Finally, by part (ii) of Theorem 2, we have $h = g_b$.

Remark 2.
1. If prime p is chosen to be sufficiently large, the possibility of the PGCD failing is low.
2. If PGCD tries to invert a zero-divisor in $\bar{L}_p$, we abort PGCD and return control to MGCD and choose a new prime.
3. As we do not know $\mathrm{lm}(g)$ in advance, there is a question as to how we can detect unlucky evaluation points. We only keep images g_i with the least $\mathrm{lm}(g_i)$ and discard the others. See lines 24 to 29 of Algorithm 2, PGCD.
4. Although lc-bad evaluation points can be ruled out in advance, we cannot detect zero-divisor or unlucky evaluation points beforehand. Therefore, we will end up calling the monic Euclidean algorithm in $\bar{L}_p[x_1]$ with zero-divisor, unlucky, and good evaluation points.

Example 8. Let $g = (6z+3)(y+2)x$, $f_1 = g \cdot (x+z+1)$, and $f_2 = g \cdot (x+2y+z+10)$ be two polynomials in $\bar{L}_{11}[x, y]$ listed in the lexicographic order with $x > y$ where $\bar{L}_{11} = \mathbb{Z}_{11}[z]/\langle z^2 + 8 \rangle$. By inspection, we can see that the monic $\gcd(f_1, f_2) = (y+2)x$. In this example, $y = 9$ is an lc-bad evaluation point, $y = 1$ is an unlucky evaluation point, and $y = 0$ is a zero-divisor evaluation point since $z^2 + 8 \mod 11 = (z+6)(z+5)$ and $\mathrm{lc}(f_1(x, 0)) = z + 6$.

Let f be a polynomial in $\bar{L}_p[x_1, \ldots, x_k]$. Let $Xk = [x_1, \ldots, x_{k-1}]$. The **content** of f w.r.t Xk, denoted by $\mathrm{cont}(f, Xk)$ is the monic gcd of coefficients of f in Xk which is a polynomial in $\bar{L}_p[x_k]$. The **primitive part** of f, w.r.t X, is defined as $\mathrm{pp}(f, Xk) = f/\mathrm{cont}(f, Xk)$. PGCD uses the property

$$\gcd(f_1, f_2) = \gcd(\mathrm{cont}(f_1, Xk), \mathrm{cont}(f_2, Xk)) \cdot \gcd(\mathrm{pp}(f_1, Xk), \mathrm{pp}(f_2, Xk)).$$

Algorithm 2: PGCD

Input: $f_1, f_2 \in \bar{L}_p[x_1, \ldots, x_k]$
Output: $\gcd(f_1, f_2) \in \bar{L}_p[x_1, \ldots, x_k]$ or FAIL

1 $Xk := [x_1, \ldots, x_{k-1}]$ $prod := 1$ **if** $k = 1$ **then**
2 $H := \gcd(f_1, f_2) \in \bar{L}_p[x_1]$ **return**(H)
3 $c := \gcd(\mathrm{cont}(f_1, Xk), \mathrm{cont}(f_2, Xk)) \in \bar{L}_p[x_k]$ **if** $c = FAIL$ **then**
4 **return**(FAIL)
5 $f_{1_p} = \mathrm{pp}(f_1, Xk)$ and $f_{2_p} = \mathrm{pp}(f_2, Xk)$ **if** $f_{1_p} = FAIL$ *or* $f_{2_p} = FAIL$ **then**
6 **return**(FAIL)
7 $\Gamma := \gcd(lc(f_{1_p}, Xk), lc(f_{2_p}, Xk)) \in \bar{L}_p[x_k]$ **if** $\Gamma = FAIL$ **then**
8 **return**(FAIL)
9 **while** *true* **do**
10 Take a new random evaluation point, $j \in \mathbb{Z}_p$, which is not lc-bad.
$F_{1_j} := f_{1_p}(x_1, \ldots, x_{k-1}, x_k = j)$ and $F_{2_j} := f_{2_p}(x_1, \ldots, x_{k-1}, x_k = j)$
$G_j := PGCD(F_{1_j}, F_{2_j}, p) \in \bar{L}_p[x_1, \ldots, x_{k-1}]$ // $\mathrm{lc}(G_j) = 1$ in lex order with $x_1 > x_2 > \ldots > x_{k-1}$
11 **if** $G_j = FAIL$ **then**
12 return(FAIL)
13 $lm := \mathrm{lm}(G_j, Xk)$ // in lex order with $x_1 > x_2 > \ldots > x_{k-1}$
14 $\Gamma_j := \Gamma(j) \in \mathbb{Z}_p$
15 $g_j := \Gamma_j \cdot G_j$ // Solve the leading coefficient problem
16 **if** $prod = 1$ *or* $lm < least$ **then**
// First iteration or all the previous evaluation points were unlucky.
17 $least,\ H,\ prod := lm,\ g_j,\ x_k - j$
18 **else**
19 **if** $lm > least$ **then**
// j is an unlucky evaluation point
20 Go back to step 12.
21 **else if** $lm = least$ **then**
// Interpolate x_k in the gcd H incrementally
22 $V_j := prod(x_k = j)^{-1} \cdot (g_j - H(x_k = j))$ $H := H + V_j \cdot prod$
$prod := prod \cdot (x_k - j)$
23 **if** $\deg(prod, x_k) > \deg(H, x_k) + 1$ **then**
24 $H := \mathrm{pp}(H, Xk)$
// Test if H is the gcd of f_1 and f_2.
25 Choose $b_2, \ldots, b_k \in \mathbb{Z}_p$ at random such that $\mathrm{lc}(H)(x_1, b_2, \ldots, b_k) \neq 0$
$A,\ B,\ C := f_1(x_1, b_2, \ldots, b_k), f_2(x_1, b_2, \ldots, b_k), H(x_1, b_2, \ldots, b_k)$ **if** $C \mid A$ *and* $C \mid B$ **then**
26 **return**$(c \cdot H)$

For $k > 1$ algorithm PGCD recursively computes monic images of the gcd in $\bar{L}_p[x_1, \ldots, x_{k-1}]$. Let $\beta_1, \ldots, \beta_j \in \mathbb{Z}_p$ be the evaluation points chosen by PGCD.

To recover the leading coefficient of g in x_k, we follow Brown [2] and scale by $\Gamma(x_k) = \gcd(\mathrm{lc}(f_1, Xk), \mathrm{lc}(f_2, Xk))$ evaluated at the current evaluation point $x_k = \beta_j$. Thus, after interpolating the gcd H we have $\mathrm{lc}(H, Xk) = \Gamma(x_k)$.

The interpolation of x_k in PGCD lines 27–29 is based on the Newton form for H, namely, $H = V_1 + V_2(x_k - \beta_1) + \cdots + V_j \prod_{i=1}^{j-1}(x_k - \beta_i)$ where $V_i \in \bar{L}_p[x_1, \ldots, x_{k-1}]$ for $1 \le i \le j$. To compute the new H from the previous H we need only compute V_j.

In the final phase of PGCD, we need to verify whether the primitive part of H is the gcd of $\mathrm{pp}(f_1, Xk)$ and $\mathrm{pp}(f_2, Xk)$. To do this, we reduce the polynomials f_1, f_2, and H to univariate polynomials in $\bar{L}_p[x_1]$ by evaluating them at $x_2 = b_2, \ldots, x_k = b_k$, where $b_2, \ldots, b_k$ are chosen at random from $\mathbb{Z}_p$ until $\mathrm{lc}(H)(x_1, b_1, \ldots, b_k) \neq 0$. Then, we check if the evaluated H divides the evaluated f_1 and f_2. If this is the case, then H is the gcd of f_1 and f_2 with high probability. Hence, PGCD is a Monte Carlo algorithm. Alternatively, if we do the division test in $\bar{L}_p[x_1, \ldots, x_k]$ rather than in $\bar{L}_p[x_1]$, then PGCD would be a Las Vegas algorithm. However, in this case, the complexity of PGCD would be dominated by the cost of the divisions in $\bar{L}_p[x_1, \ldots, x_k]$.

3.2 MGCD

The MGCD algorithm, as presented in Algorithm 3, is a Monte Carlo algorithm for computing the monic $g = \gcd(f_1, f_2)$ where $f_1, f_2 \in L[x_1, \ldots, x_k]$. MGCD begins with a preprocessing step where the input polynomials, f_1, f_2, and the minimal polynomials $M_1, \ldots, M_n$ are replaced with their semi-associates. Let ϕ_p denote the modular homomorphism, that is, $\phi_p(f) = f \mod p$. MGCD chooses a prime p and applies ϕ_p to map the coefficients in L to L_p. Subsequently, it employs the isomorphism ϕ_γ to convert the polynomials over L_p to their corresponding polynomials over $\bar{L}_p$. Then MGCD calls PGCD to find the monic gcd in $\bar{L}_p[x_1, \ldots, x_k]$. Let G_p be the output of PGCD. If $G_p = FAIL$, either p is a zero-divisor prime or the PGCD algorithm encounters a zero-divisor evaluation point. In both cases, the algorithm goes back to step 4 to choose a new prime. In step 14, $G_p \in \bar{L}_p[x_1, \ldots, x_k]$ will be converted to its corresponding polynomial over L_p. Applying Theorem 2, MGCD just keeps the gcd images G_p with the least leading monomial for Chinese remaindering. For instance, if G_{p_i} is the output of PGCD at the ith iteration, and if $\mathrm{lm}(G_{p_i}) > \mathrm{lm}(G_{p_{i-1}})$, then p_i is an unlucky prime and we simply ignore its result G_{p_i} and choose another prime.

After Chinese remaindering, MGCD employs rational number reconstruction (RNR) [9,18] to recover the coefficients of the potential gcd in L. Failure in the RNR call means the product of the primes is not large enough to recover the rational coefficients. If RNR does not fail, then we follow the same strategy as in PGCD to verify if H could be the gcd of f_1 and f_2 or not.

Remark 3. For the efficiency of the MGCD algorithm, it is necessary to apply ϕ_p before ϕ_γ. This eliminates expression swell in $\mathbb{Q}$.

Algorithm 3: MGCD

Input: $f_1, f_2 \in L[x_1, \ldots, x_k]$ where $L = \mathbb{Q}[z_1, \ldots, z_n]/\langle M_1(z_1), \ldots, M_n(z_n)\rangle$
Output: $\gcd(f_1, f_2)$

$M := 1$
$f_1 := \check{f}_1$ and $f_2 := \check{f}_2$ // Clear fractions
while *true* **do**
  Choose a new random prime p, that is not, lc-bad.
  Choose $C_1, \ldots, C_{n-1} \in [1, p)$ at random and set $\gamma = z_1 + \sum_{i=2}^{n} C_{i-1} z_i$
  Call Algorithm 1 with inputs $[\phi_p(\check{M}_1), \ldots, \phi_p(\check{M}_n)]$, $\mathbb{Z}_p$ and $\phi_p(\gamma)$ to compute $M(z)$, A, and A^{-1}
  if *Algorithm 1 fails* **then**
    Go back to step 4
  // Apply Algorithm 2 to get the monic gcd over $\bar{L}_p$
  $G_p = PGCD(\phi_\gamma(\phi_p(f_1)), \phi_\gamma(\phi_p(f_2))) \in \bar{L}_p[x_1, \ldots, x_k]$
  if $G_p =$ *FAIL* **then**
    // p is a zero-divisor prime or PGCD has encountered a zero-divisor evaluation point.
    Go back to step 4.
  if $\deg(G_p) = 0$ **then**
    return(1)
  // Convert $G_p \in \bar{L}_p$ to its corresponding polynomial over L_p
  $G_p := \phi_\gamma^{-1}(G_p)$
  $lm := \mathrm{lm}(G_p)$ w.r.t lexicographic order with $x_1 > x_2 \ldots > x_k$
  if $M = 1$ *or* $lm < least$ // First iteration or all the previous primes were unlucky.
  then
    $G, least, M := G_p, lm, p$
  else
    if $lm = least$ **then**
      Using CRT, compute $G' \equiv G \mod M$ and $G' \equiv G_p \mod p$
      set $G = G'$ and $M = M \cdot p$
    else if $lm > least$ **then**
      // p is an unlucky prime
      Go back to step 4
  $H :=$ Rational Number Reconstruction of G mod M
  if $H \neq$ *FAIL* **then**
    Choose a new prime q and $b_2, \ldots, b_n \in \mathbb{Z}_q$ at random such that $\mathrm{lc}(H)(x_1, b_2, \ldots, b_k) \neq 0$
    $A,\ B,\ C := f_1(x_1, b_2, \ldots, b_k), f_2(x_1, b_2, \ldots, b_k), H(x_1, b_2, \ldots, b_k)$
    // A, B, C are polynomials in $L_q[x_1]$
    if $C \mid A$ *and* $C \mid B$ **then**
      return(H)

In step 4 of the Algorithm 3, we choose a prime to reduce inputs modulo it. However, not all the primes result in the successful reconstruction of the monic gcd. We distinguish five types of primes in the following definition.

Definition 3. *Let $f_1, f_2 \in L[x_1, \ldots, x_k]$ and p be a prime. We distinguish the following cases:*

- ***Lc-bad Prime:*** *If p divides $\mathrm{lc}(\check{f}_1)$ or any $\mathrm{lc}(\check{M}_1(z_1)), \ldots, \mathrm{lc}(\check{M}_n(z_n))$, then we call p an lc-bad prime.*
- ***Det-bad Prime:*** *If $\det(A) \mod p = 0$, where A is the coefficient matrix of powers of γ obtained from the LAminpoly algorithm, then p is called a* det*-bad prime.*
- ***Zero-Divisor Prime:*** *If p is neither an lc-bad nor a* det*-bad prime and the PGCD algorithm fails for p, in steps $4, 6, 8, 10, 31$, then p is called a zero-divisor prime.*
- ***Unlucky Prime:*** *Let $g_p = \gcd(\phi_p(\check{f}_1), \phi_p(\check{f}_2))$. If $\mathrm{lm}(g_p) > \mathrm{lm}(\gcd(f_1, f_2))$, then we call p an unlucky prime. Considering Theorem 2, the results of these primes must be ignored.*
- ***Good Prime:*** *If prime p is not an lc-bad, det-bad, unlucky, or zero-divisor prime, we define it as a good prime.*

Theorem 4. *Let $f_1, f_2 \in L[x_1, \ldots, x_k]$ and g be the monic $\gcd(f_1, f_2)$. If p is a good prime and the monic $\gcd(\phi_p(f_1), \phi_p(f_2))$, g_p, exists, then $g_p = \phi_p(g)$.*

Proof. If p is good then p is not lc-bad so we may apply Theorem 2 with $R = L$ and $R' = L_p$ so $\mathrm{lm}(g_p) \geq lm(g)$. But p is not unlucky so $\mathrm{lm}(g_p) = \mathrm{lm}(g)$. By part (ii) of Theorem 2 we have $g_p = \phi_p(g)$ as required.

Example 9. Let $L = \mathbb{Q}[z, w]/\langle z^2 - 2, w^2 - 3\rangle$, and $f_1 = (x+w)(5x+2w+z)xw$ and $f_2 = (x + w)(5x + 9w + z)$ be polynomials in $L[x]$. By inspection, $\gcd(f_1, f_2) = (x + w)$. In this example $p = 5$ is an lc-bad prime, $p = 7$ is an unlucky prime, and $p = 3$ is a zero-divisor prime since $w^2 - 3 \mod 3 = w^2$.

4 Complexity

Let $H(f)$ denote the **height** of $f \in L[x_1, \ldots, x_k]$ which is the magnitude of the largest integer coefficient of $\check{f}$. Let $\#f$ denote the number of terms of f. Let $f_1, f_2 \in L[x_1, \cdots, x_k]$ and g be the monic $\gcd(f_1, f_2)$. The quantities involved in the running time of the MGCD algorithm are as follows:

- N is the number of good primes needed to reconstruct the monic gcd g
- $T_f = \max(\#f_1, \#f_2)$ and $T_g = \#g$
- $M = \log \max_{i=1}^{n} H(\check{m}_i)$ and $C = \log \max(H(\check{f}_1), H(\check{f}_2))$.
- $D = \max_{i=1}^{k} \max(\deg(f_1, x_i), \deg(f_2, x_i))$ and $d = [L : \mathbb{Q}]$.

We assume that multiplication and inverses in $\bar{L}_p$ cost $O(d^2)$ as our implementation currently uses classical quadratic polynomial arithmetic.

Theorem 5. *The expected time complexity of our MGCD algorithm is*

$$O(N(M + CT_f)d + Nd^2(d + T_f + T_g) + Nd^2D^{k+1} + N^2dT_g)$$

Proof. In the MGCD algorithm, the most dominant operations are as follows:

1. **Modular homomorphism:** The MGCD algorithm reduces the minimal polynomials $\check{M}_1, \ldots, \check{M}_n$ and the input polynomials $\check{f}_1$ and $\check{f}_2$ mod a prime. For N primes this costs $O(N(M + CT_f)d)$.
2. ϕ_γ **isomorphism:** The time complexity of building the matrix A is $O(d^3)$, and the running time complexity of applying ϕ_γ to the T_f non-zero terms of f_1 and f_2 for N primes is $O(Nd^2T_f)$. Additionally, let G_p be the output of the PGCD algorithm in step 9. The time complexity of calling ϕ_γ^{-1} for G_p in step 14 for N primes is $O(Nd^2T_g)$.
3. **PGCD:** Brown's PGCD algorithm [2] does $O(D^{k+1})$ arithmetic operations in $\mathbb{Z}_p$. Accordingly, our PGCD algorithm does $O(D^{k+1})$ arithmetic operations in $\bar{L}_p$ each of which costs $O(d^2)$. Overall, our PGCD costs $O(d^2D^{k+1})$. The dominating step is the $O(D^{k-1})$ calls to the monic Euclidean algorithm in $\bar{L}_p[x_1]$ each of which does $O(D^2)$ arithmetic operations in $\bar{L}_p$.
4. **CRT and RNR:** Reconstructing $O(dT_g)$ rational coefficients in step 21 and 25 costs $O(N^2)$ each hence $O(N^2dT_g)$ in total.

The theorem follows by adding the four costs explained above.

Remark 4. Theorem 5 describes the cost of our implementation of algorithms MGCD and PGCD. We are currently working on replacing Brown's dense interpolation with a sparse interpolation approach. In the case where we interpolate g (when $\mathrm{lc}(g, x_1) = \gcd(\mathrm{lc}(f_1, x_1), \mathrm{lc}(f_2, x_1))$), the number of calls to the monic Euclidean algorithm in $\bar{L}_p[x_1]$ is reduced from $O(D^{k-1})$ to $O(kDT_g)$ using Zippel's algorithm from [19] and $O(T_g)$ using Hu and Monagan's algorithm [6]. The latter is based on the work of Ben-Or and Tiwari [1] and others.

5 Implementation

We have implemented algorithms MGCD and PGCD in Maple [10]. We use the recursive dense data structure from [17] to represent elements of $L = \mathbb{Q}(\alpha_1, \cdots, \alpha_n)$ and polynomials in $L[x_1, \ldots, x_k]$. See Fateman [5] for a comparison of the recursive dense data structure with other sparse polynomial data structures. The `rpoly` command below converts from Maple's polynomial representation to the recursive dense representation. For usability, this representation is automatically converted back to Maple's polynomial representation for display. In Maple $[1, 2, 3]$ is a Maple lists which is a read only array.

```
> f:=rpoly(2*x^2+3*x*y^2,[x,y]);
```

$$f := 2x^2 + 3xy^2$$

```
> lprint(f);  # print the actual data value
```

$$POLYNOMIAL(\,[0,[x,y],[]],\ [0,[0,0,3],[2]]\,)$$

As it is shown, the POLYNOMIAL data structure has two fields.

- The first, $[0,[x,y],[\,]]$, is the ring. The first entry, 0, indicates the characteristic of the ring. The second entry, $[x,y]$, is the list of variables. The third entry $[\,]$, which is an empty list, indicates that there are no extensions.
- The second field, $[0,[0,0,3],[2]]$, represents the polynomial recursively. To do so, it uses the fact that $\mathbb{Q}[x_1,\ldots,x_k] \cong \mathbb{Q}[x_k][x_{k-1}]\ldots[x_1]$. In this example, x is the main variable and `rpoly` maps $f \in \mathbb{Q}[x,y]$ to $f := 2x^2 + (3y^2)x \in \mathbb{Q}[y][x]$. Consequently, the entries $0,[0,0,3],[2]$ are the coefficients of x^0, x^1, and x^2, and correspond to $0, 0+0\cdot y+3y^2$, and 2 respectively.

Example 10. Let $L = \mathbb{Q}[z,w]/\langle z^2-2, w^2-3\rangle$ and $f = 2x^3+3xy^2-5wz+4$. In the following we construct the field of L, the polynomial $f \in L[x,y]$, and compute $\phi_7(f)(x,2)$.

```
> L:=rring([z,w],[z^2-2,w^2-3]);  # L=Q[z,w]/<z^2-2,w^2-3>
```

$$L := [0,[z,w],[[[-2],0,[1]],[-3,0,1]]]$$

```
> Lxy:=rring(L,[x,y]);  # Construct L[x,y] from L
```

$$Lxy := [0,[x,y,z,w],[[[-2],0,[1]],[-3,0,1]]]$$

```
> f:=rpoly(2*x^3+3*x*y^2-5*z*w+2*z^2,Lxy);
```

$$f := 2x^3+3xy^2-5wz+4 \mod < z^2-2, w^2-3 >$$

```
> getpoly(f);  # The recursive dense representation of f
```

$$[[[[4],[0,-5]]],[0,0,[[3]]],0,[[[2]]]]$$

```
> g := phirpoly(f,7);  # Apply the modular homomorphism with p = 7
```

$$g := 2x^3+3xy^2+2wz+4 \mod \langle z^2+5, w^2+4, 7\rangle$$

```
> h := evalrpoly(g,y=2);
```

$$h := 2x^3+2wz+5x+4 \mod \langle z^2+5, w^2+4, 7\rangle$$

```
> getring(h);  # The ring Lp[x]
```

$$[7,[x,z,w],[[[5],0,[1]],[4,0,1]]]$$

```
> getpoly(h);
```

$$[[[4],[0,2]],[[5]],0,[[2]]]$$

5.1 Maple Implementation

In this section, we demonstrate an application of our MGCD algorithm. First, we construct the field of $L = \mathbb{Q}[z, w]/\langle z^2 - 2, w^2 - 3\rangle$. Then, we convert two polynomials f_1 and f_2 from Maple's native representation to the recursive dense representation and compute their gcd using MGCD. MGCD prints all the used primes, lc-bad, zero-divisor, unlucky, and det-bad primes. We tell MGCD to start with a very small prime, 5, for illustrative purposes only. By default MGCD uses 31 bit primes. This is because for polynomial arithmetic in $\mathbb{Z}_p[x]$, Maple uses hardware integer arithmetic for $\mathbb{Z}_p$ for primes less than $2^{31.5}$, otherwise Maple uses GMP's multi-precision integer arithmetic which is a lot slower.

```
> L:=rring([z,w],[z^2-2,w^2-3]):
> Lxy:=rring(L,[x,y]):
> f1:=rpoly((w+5)*(x+y+w)*(14*x+2*w+z),Lxy);
```

$$f1 := (14w + 70)\,x^2 + ((14w + 70)\,y + (w + 5)\,z + 80w + 48)\,x + ((w + 5)\,z + 10w + 6)\,y + (5w + 3)\,z + 6w + 30 \mod \langle w^2 - 3, z^2 - 2\rangle$$

```
> f2:=rpoly((x+y+w)*(x+2*w+z),Lxy);
```

$$f2 := x^2 + (y + 3w + z)\,x + (2w + z)\,y + zw + 6 \mod \langle w^2 - 3, z^2 - 2\rangle$$

```
> mgcd:=MGCD(f1,f2,5);

MGCD:prime=5
gamma:=4*w+z and M(z)=z^4+1
p=5 is a ZD prime ZD=z^2+3
MGCD:prime=7
p=7 is an lc-bad prime
MGCD:prime=11
gamma:=3*w+z and M(z)=z^4+8*z^2+9
p=11 is a ZD prime ZD=z^2+8*z+3
MGCD:prime=13
gamma:=z+10*w and M(z)=z^4+7*z^2+1
MGCD:prime=17
gamma:=z+13*w and M(z)=z^4+2*z^2+8
p=17 and All the previous primes were unlucky
MGCD:prime=19
gamma:=z+17*w and M(z)=z^4+10*z^2+5
```

$$mgcd := x + y + w \mod \langle z^2 - 2, w^2 - 3\rangle$$

5.2 Benchmark

We give one benchmark for gcd computations in $L[x, y]$ where the number field $L = \mathbb{Q}(\sqrt{2}, \sqrt{3}, \sqrt{5}, \sqrt{7}, \sqrt{11})$ has degree 32. In the Table 1, the input polynomials f_1 and f_2 have degree d in x and y and their gcd g has degree 2 in x and y.

Table 1. Computation timings in CPU seconds for gcds in the ring $L[x, y]$ where $L = \mathbb{Q}(\sqrt{2}, \sqrt{3}, \sqrt{5}, \sqrt{7}, \sqrt{11})$.

d	New MGCD			Old MGCD	
	time	LAMP	PGCD	time	PGCD
4	0.119	0.023	0.027	0.114	0.100
6	0.137	0.016	0.034	0.184	0.156
8	0.217	0.018	0.045	0.330	0.244
10	0.252	0.018	0.087	0.479	0.400
12	0.352	0.018	0.078	0.714	0.511
16	0.599	0.017	0.129	1.244	1.008
20	0.767	0.017	0.161	1.965	1.643
24	1.103	0.019	0.220	2.896	2.342
28	1.890	0.023	0.358	4.487	3.897
32	2.002	0.020	0.392	5.416	4.454
36	2.461	0.017	0.595	6.944	5.883
40	3.298	0.019	0.772	9.492	7.960

Column New MGCD is the time for our new algorithm using a primitive element and computing over $\bar{L}_p$. Column Old MGCD is the time for MGCD if we do not use a primitive element and compute over L_p. Column LAMP is the time spent in Algorithm LAminpoly. For both algorithms, column PGCD is the time spent in Algorithm PGCD. The speedup gained by using ϕ_γ is seen by comparing columns PGCD. These preliminary timings show a speedup of PGCD of a factor of 10 which is promising. The benchmark was run on an Intel Gold 6342 CPU running at 2.8 GHz. We used Maple 2022. For the details of the benchmark, see http://www.cecm.sfu.ca/~mmonagan/code/MGCD .

6 Conclusion and Future Work

Let $f_1, f_2 \in \mathbb{Q}(\alpha_1, \cdots, \alpha_n)[x_1, \ldots, x_n]$, and let g be their monic gcd. We have designed a multivariate modular gcd algorithm, MGCD, to compute g. For each prime p chosen by MGCD, to speed up the coefficient arithmetic in $\mathbb{Q}(\alpha_1, \cdots, \alpha_n)$ mod p, we use a primitive element γ modulo p.

For future work, we need to compute the probability that MGCD obtains an incorrect answer, as well as the probabilities of getting unlucky, zero-divisor, lc-bad primes, and evaluation points. For arithmetic in $\bar{L}_p = \mathbb{Z}_p[z]/\langle M(z)\rangle$, our Maple implementation currently uses classical $O(d^2)$ algorithms where $d = \deg(M)$. For large d, we can speed up multiplication in $\bar{L}_p$ by using fast multiplication and division for $\mathbb{Z}_p[z]$.

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

References

1. Ben-Or, M., Tiwari, P.: A deterministic algorithm for sparse multivariate polynomial interpolation. In: Proceedings of STOC 1988, pp. 301–309. ACM (1988)
2. Brown, W.S.: On Euclid's algorithm and the computation of polynomial greatest common divisors. J. ACM **18**, 478–504 (1971)
3. Collins, G.E.: Subresultants and reduced polynomial remainder sequences. J. ACM **14**, 128–142 (1967)
4. Encarnación, M.J.: Computing GCDs of polynomials over algebraic number fields. J. Symb. Comput. **20**, 299–313 (1995)
5. Fateman, R.: Comparing the speed of programs for sparse polynomial multiplication. SIGSAM Bull. **37**(1), 4–15 (2003)
6. Jiaxiong, H., Monagan, M.: A fast parallel sparse polynomial GCD algorithm. Symb. Comput. **105**(1), 28–63 (2021)
7. Langemyr, L., McCallum, S.: The computation of polynomial greatest common divisors over an algebraic number field. J. Symb. Comput. **8**(5), 429–448 (1989)
8. Lin, X., Maza, M.M., Schost, É.: Fast arithmetic for triangular sets: from theory to practice. J. Symb. Comput. **44**(7), 891–907 (2009)
9. Monagan, M.: Maximal quotient rational reconstruction: an almost optimal algorithm for rational reconstruction. In: Proceedings of ISSAC 2004, pp. 243–249. ACM (2004)
10. Monagan, M., et al.: Maple 8 Introductory Programming Guide (2003)
11. Poteaux, A., Schost, É.: Modular composition modulo triangular sets and applications. Comput. Complex. **22**, 463–516 (2013)
12. Poteaux, A., Schost, É.: On the complexity of computing with zero-dimensional triangular sets. J. Symb. Comput. **50**, 110–138 (2013)
13. Smedley, T.: A new modular algorithm for computation of algebraic number polynomial GCDs. In: Proceedings of ISSAC 1989, pp. 91–94. ACM (1989)
14. Trager, B.M.: Algebraic factoring and rational function integration. In: Proceedings of the Third ACM Symposium on Symbolic and Algebraic Computation, SYMSAC 1976, pp. 219–226. ACM (1976)
15. van der Hoeven, J., Lecerf, G.: Accelerated tower arithmetic. J. Complex. **55**, 101402 (2019)
16. van der Hoeven, J., Lecerf, G.: Directed evaluation. J. Complex. **60**, 101498 (2020)
17. van Hoeij, M., Monagan, M.: A modular GCD algorithm over number fields presented with multiple extensions. In: Proceedings of the 2002 International Symposium on Symbolic and Algebraic Computation, ISSAC 2002, pp. 109–116. ACM (2002)

18. Wang, P., Guy, M.J.T., Davenport, J.H.: P-adic reconstruction of rational numbers. SIGSAM Bull. **16**(2), 2–3 (1982)
19. Zippel, R.: Probabilistic algorithms for sparse polynomials. In: Ng, E.W. (ed.) Symbolic and Algebraic Computation. LNCS, vol. 72, pp. 216–226. Springer, Heidelberg (1979). https://doi.org/10.1007/3-540-09519-5_73