Computing the Greatest Common Divisor of Multivariate Polynomials over Finite Fields

Suling Yang
Simon Fraser University
syangc@cecm.sfu.ca

Abstract

Richard Zippel's sparse modular GCD algorithm is widely used to compute the monic greatest common divisor (GCD) of two multivariate polynomials over $\mathbb{Z}$. In this report, we present how this algorithm can be modified to solve the GCD problem for polynomials over finite fields of small cardinality. When the GCD is not monic, Zippel's algorithm cannot be applied unless the normalization problem is resolved. In [6], Alan Wittkopf *et al.* developed the LINZIP algorithm for solving the normalization problem. Mahdi Javadi proposed a refinement to the LINZIP algorithm in [4]. We implemented his approach and will show that it is efficient and effective on polynomials over small finite fields. Zippel's algorithm also uses properties of transposed Vandermonde systems to reduce the time and space complexity of his algorithm. We also investigated how this can be applied to our case.

1 Introduction

Let A and B be polynomials in $\mathcal{F}[x_1, \ldots, x_n]$, where $\mathcal{F}$ is a *unique factorization domain* (UFD) and n is a positive integer. Our goal is to find a *greatest common divisor* (GCD) G of A and B. Let $\bar{A} = A/G$, $\bar{B} = B/G$ be the cofactors of A and B, respectively. When $n = 1$, A and B are univariate, and so is G. In this case the Euclidean algorithm can be used. However, we want to develop efficient algorithms for computing multivariate GCD,

because the size of coefficients grows rapidly in $\mathcal{F}[x_1, \ldots, x_n]$ when using the primitive Euclidean algorithm [2]. This problem is similar to the growth in the size of coefficients when using naive methods to solve linear equations over $\mathcal{F}[x_1, \ldots, x_n]$. Using homomorphisms to map the GCD problem to a simpler domain can improve arithmetic calculation and lead to better approaches to avoid the problems with coefficient growth.

In [1], Brown developed a modular algorithm which consists of two procedures to find the GCD of A and B when A and B are polynomials with integer coefficients $\mathbb{Z}$. Brown's algorithm finds the GCD's images in univariate domain, and then uses Chinese Remainder Theorem (CRT) or Polynomial Interpolation to get the original GCD. The running time of Brown's algorithm depends on the total degree of G. In [5] Erich Kaltofen and Michael Monagan explored the generic setting of the modular GCD algorithm. They showed how it could be applied to GCD problem over the Euclidean ring $\mathbb{Z}/(p)[t]$, where $\mathbb{Z}/(p)$ denotes the integer residues modulo p. They also compared it with other algorithms in the domain $\mathbb{Z}/(p)[t][x]$ and established it as better than the known other standard methods.

Zippel presented a sparse modular algorithm which improved the complexity of Brown's algorithm for sparse G, by reducing the number of univariate images required in [7]. Zippel's algorithm obtains an assumed form of G by recursively applying the algorithm in one fewer variable, and then calculates the constant coefficients from univariate GCD images (we call this *Sparse Interpolation*). It is a *Las Vegas* algorithm which always results in a correct solution or declares failure. Sparse interpolation requires $n_{max} + 1$ images where n_{max} is the maximum number of terms of the coefficients of the main variable x_1 in G. In [8] Zippel analyzed the properties of *Vandermonde matrices*, whose entries are powers of a same constant in each column and same power in each row. Zippel constructed an algorithm to find the inverse of a Vandermonde matrix using linear space and quadratic time. We call it *LinSpaceSol* algorithm. We will discuss these properties of Vandermonde matrices in the next section.

If the GCD G is non-monic in the main variable x_1, for example, $G = (x_2^5 + x_3)x_1^8 + (x_2 + x_3)x_1^2 + (x_2 + x_3^2 + 1) \in \mathbb{Z}[x_2, x_3][x_1]$, then Zippel's sparse

modular algorithm cannot be applied directly. This is called the *normalization problem*. In order to solve this problem efficiently, de Kleine, Monagan and Wittkopf implemented the LINZIP algorithm which uses $\mathcal{O}(n_1^3 + n_2^3 + \cdots + n_t^3)$ time, where t is the number of terms in G when it is written in the collected form in the main variable x_1, and n_i's are the numbers of terms in the coefficients of x_1. E.g. for the above example, $t = 3$, $n_1 = 2$, $n_2 = 2$, and $n_3 = 3$. In [4], M. Javadi developed a more efficient method to solve the normalization problem using only $\mathcal{O}((n_1 + \cdots + n_k)^3 + n_{k+1}^2 + \cdots + n_t^2)$ time, where $n_1 \leq n_2 \leq \cdots \leq n_t$ and k is the number of coefficients of x_1 that the algorithm needs to compute the scaling factor.

In the following section, we discuss the work from previous authors. Then, the algorithms of solving the GCD problems over finite fields of small cardinalities are given in Section 3. In Section 4, we analyze the probability of getting an incorrect result, and thus triggering a restart. Experimental results of the effectiveness and efficiency are given in Section 5. Section 6 concludes this report by providing some potential improvement.

2 Related Work

2.1 Modular GCD Algorithm

Consider $A, B \in \mathbb{Z}[x_1, \cdots, x_n]$. Brown's algorithm for solving the GCD problem in $\mathbb{Z}[x_1, \cdots, x_n]$ consists two algorithms, MGCD and PGCD, to deal with different types of homomorphisms in $\mathbb{Z}[x_1, \cdots, x_n]$. The MGCD algorithm reduces a GCD problem to a series of GCD problems in $\mathbb{Z}_{p_i}[x_1, \cdots, x_n]$ by applying modular homomorphisms [1]. It chooses a sequence of primes $p_i \in \mathbb{Z}$, such that p_i does not divide $LC_X(A), LC_X(B)$ where $LC_X(A)$ means the leading coefficient of A in the lexicographic order of $X = [x_1, \cdots, x_n]$, and repeatedly calls PGCD to obtain $G_i = gcd(A \mod p_i, B \mod p_i)$. It applies the *Chinese Remainder Theorem* (CRT) on all G_i's with moduli p_i's incrementally and the stabilized image is G if it divides both A and B.

Similarly, the PGCD algorithm reduces the $\mathbb{Z}_{p_i}[x_1, \cdots, x_n]$ GCD problem to a series of $(n-2)$-variate finite field GCD problems in $\mathbb{Z}_{p_i}[x_1, \cdots, x_{n-1}]$

by applying evaluation homomorphisms. It chooses $\alpha_j \in \mathbb{Z}_{p_i}$ such that $LC_{\hat{X}}(A)(x_{n-1} = \alpha_j), LC_{\hat{X}}(B)(x_{n-1} = \alpha_j) \neq 0$ where $\hat{X} = [x_1, \cdots, x_{n-1}]$. Then it recursively calls itself to obtain $G_{i,j} = gcd((A \mod p_i)(x_{n-1} = \alpha_j), (B \mod p_i)(x_{n-1} = \alpha_j))$. Then, it applies *polynomial interpolation* on coefficients of $G_{i,j}$ to interpolate x_{n-1} incrementally stopping when the interpolated result stabilizes, and the stabilized image is $G_i = gcd(A \mod p_i, B \mod p_i)$ if it divides both $A \mod p_i$ and $B \mod p_i$ in $\mathbb{Z}_{p_i}[x_1, \cdots, x_n]$.

2.2 Problems with the Modular GCD Algorithm

After the major problem is reduced to a simpler problem over a more algebraic structure domain which allows for a wider range of algorithms, the arithmetic is simpler because the arithmetic is done in a domain with small coefficients. However, the trade-off is information loss, which may result in failure in some cases. Let $G = gcd(A, B)$ where $A, B \in \mathbb{Z}[x_1, \cdots, x_n]$, and let $\bar{A} = A/G, \bar{B} = B/G$. Let $H = gcd(\phi_p(A), \phi_p(B))$ in $\mathbb{Z}_p[x_1, \cdots, x_n]$. The problem is that H may not be a scalar multiple of $\phi_p(G)$.

Definition 2.1. A homomorphism ϕ is *bad* if $\deg_{x_1}(\phi(G)) < \deg_{x_1}(G)$. If $\deg_{x_1}(\phi(G)) = \deg_{x_1}(G \mod p) < \deg_{x_1}(G)$ where p is a *prime* in $\mathbb{Z}$, then p is *bad*. Similarly, if $\deg_{x_1}(\phi(G)) = \deg_{x_1}(G \mod I) < \deg_{x_1}(G)$ where $I = < x_2 - \alpha_2, ..., x_n - \alpha_n >, \alpha_i \in \mathbb{Z}_p$, then the evaluation point $(\alpha_2, ..., \alpha_n)$ is *bad*.

Definition 2.2. A homomorphism ϕ is *unlucky* if

$$\deg_{x_1}(GCD(\phi(\bar{A}), \phi(\bar{B}))) > 0.$$

Bad homomorphisms can be prevented if we choose p and $(\alpha_2, ..., \alpha_n)$ such that $LC_{x_1}(A) \mod p \neq 0$ and $LC_{x_1}(B) \mod I \neq 0$. However, unlucky homomorphisms cannot be detected in advance. Instead, by the application of the following lemma, Brown's algorithms can identify unlucky homomorphisms at execution time.

Lemma 2.3. (Lemma 7.3 from Geddes *et al.* [2]) *Let R and R' be UFD's with $\phi : R \to R'$ a homomorphism of rings. This induces a natural homomorphism, also denoted by ϕ, from $R[x]$ to $R'[x]$. Suppose $A(x), B(x) \in R[x]$ and $G(x) = GCD(A(x), B(x))$ with $\phi(LC(G(x))) \neq 0$. Then*

$$\deg_x(GCD(\phi(A(x)), \phi(B(x)))) \geq \deg_x(GCD(A(x), B(x))). \tag{2.1}$$

Therefore, we can eliminate a univariate image of higher degree than other univariate images. The probability of getting an unlucky evaluation point is analyzed in Section 4.
Example 1:

$$A = (x + y + 11)(3x + y + 1)$$
$$B = (x + y + 4)(3x + y + 1)$$

If we choose $p = 3$, then $gcd(A \mod 3, B \mod 3) = y + 1$, which has degree 0 in x. Thus, $p = 3$ is bad. If we choose $p = 7$, then $gcd(A \mod 3, B \mod 3) = (x + y + 4)(3x + y + 1)$. Thus, $p = 7$ is unlucky.

2.3 Sparse Modular GCD Algorithm

We have seen in the previous sections that modular GCD algorithms can solve problems by reducing a complex problem into a number of easier problems. However, another problem that arises from this approach is the growth in the number of univariate GCD images required. In many cases, especially when polynomials are multivariate, G is sparse, i.e., the number of nonzero terms is generally much smaller than the number of possible terms up to a given total degree d. Hence, sparse algorithms may be more efficient. Zippel introduced a sparse algorithm for calculating the GCD of two multivariate polynomials over the integer [7]. We will show the pseudo-code of the algorithm applied on finite fields in the next section. This approach is probabilistic, and we will estimate the likelihood of success in the Section 4.

One observation is that if an evaluation point is chosen at random from a large enough set, then evaluating a polynomial at that point is rarely zero.

Based on this observation, the sparse modular methods determine a solution for one small domain by normal approach, and then use sparse interpolations to find solutions for other small domains. For a GCD problem in n variables where the actual GCD has total degree d, a dense interpolation, for instance, Newton interpolation, requires $(d+1)^n$ evaluation points, since it assumes none of the possible terms is absent. Sparse interpolation assumes that the image G_f from the dense interpolation is of correct form, and it is to determine t coefficients where t is the number of terms in G_f and $t \ll d$. Hence, it requires only $\mathcal{O}\left(n(t+1)(d+1)\right)$ evaluation points.

2.4 Vandermonde Matrices Applied to Interpolation

In the sparse interpolation, we need to find solutions for a set of linear equations over a field $\mathcal{F}$, i.e., we need to find the inverse of the matrix formed by these equations. Algorithms for finding inverses of general matrices require $\mathcal{O}(n^3)$ arithmetic operations in $\mathcal{F}$ and space for $\mathcal{O}(n^2)$ elements of $\mathcal{F}$, where n is the number of rows/columns. But for Vandermonde matrices, one can find the inverse using $\mathcal{O}(n)$ space and $\mathcal{O}(n^2)$ arithmetic operations. The form of a Vandermonde matrix is as follows.

$$V_n = \begin{bmatrix} 1 & k_1 & k_1^2 & \dots & k_1^{n-1} \\ 1 & k_2 & k_2^2 & \dots & k_2^{n-1} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & k_n & k_n^2 & \dots & k_n^{n-1} \end{bmatrix}, \tag{2.2}$$

where the k_i are chosen from $\mathcal{F}$. We can easily calculate the determinant of a Vandermonde matrix. We can observe that multiplying the ith column by k_1 and subtract it from the $i+1$th column we get,

$$\det V_n = \begin{vmatrix} 1 & 0 & 0 & \dots & 0 \\ 1 & k_2 - k_1 & k_2^2 - k_1 k_2 & \dots & k_2^{n-1} - k_1 k_2^{n-2} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & k_n - k_1 & k_n^2 - k_1 k_n & \dots & k_n^{n-1} - k_1 k_n^{n-2} \end{vmatrix}.$$

We can factor out $(k_2 - k_1)$ from the second row, $(k_3 - k_1)$ from the third row, and so on.

$$\det V_n = 1 \cdot \begin{vmatrix} (k_2-k_1)\cdot 1 & (k_2-k_1)\cdot k_2 & \dots & (k_2-k_1)\cdot k_2^{n-2} \\ (k_3-k_1)\cdot 1 & (k_3-k_1)\cdot k_3 & \dots & (k_3-k_1)\cdot k_3^{n-2} \\ \vdots & \vdots & \dots & \vdots \\ (k_n-k_1)\cdot 1 & (k_n-k_1)\cdot k_n & \dots & (k_n-k_1)\cdot k_n^{n-2} \end{vmatrix}.$$

Therefore,

$$\begin{aligned} \det\, V_n &= \prod_{1<i\le n} (k_i - k_1) \cdot \det \left(\begin{bmatrix} 1 & k_2 & k_2^2 & \dots & k_2^{n-2} \\ 1 & k_3 & k_3^2 & \dots & k_3^{n-2} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & k_n & k_n^2 & \dots & k_n^{n-2} \end{bmatrix} \right) \\ &= \prod_{1<i\le n} (k_i - k_1) \cdot \det\, V_{n-1}. \end{aligned}$$

Since we know $\det V_1 = \det([1]) = 1$, we can determine the determinant of a Vandermonde matrix by applying the above result recursively we have.

Theorem 2.4. *The determinant of the Vandermonde matrix is*

$$\det\, V_n = \prod_{1\le i<j\le n} (k_j - k_i). \tag{2.3}$$

Corollary 2.5. *The determinant of a Vandermonde matrix is non-zero if and only if the k_i are distinct.*

Assume that $V_n^{-1} = [a_{ij}]$ is the inverse of the Vandermonde matrix V_n and $\mathbf{I}$ is the identity matrix. Then,

$$V_n \cdot V_n^{-1} = \begin{bmatrix} 1 & k_1 & k_1^2 & \dots & k_1^{n-1} \\ 1 & k_2 & k_2^2 & \dots & k_2^{n-1} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ 1 & k_n & k_n^2 & \dots & k_n^{n-1} \end{bmatrix} \cdot \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1n} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2n} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ a_{n1} & a_{n2} & a_{n3} & \dots & a_{nn} \end{bmatrix} = \mathbf{I}\,.$$

Consider jth element of the ith row of the product $V_n \cdot V_n^{-1}$ as a polynomial in k_i as follows.

$$P_j(k_i) = a_{1j} + a_{2j}k_i + a_{3j}k_i^2 + \cdot + a_{nj}k_i^{n-1} \tag{2.4}$$

Then we know,

$$P_j(k_i) = \begin{cases} 1 & \text{if } i = j \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

By choosing the $P_j(Z)$ to be

$$P_j(Z) = \prod_{\substack{l \neq j \\ 1 \leq l \leq n}} \frac{Z - k_l}{k_j - k_l},$$

we can verify that equation (2.5) holds, and thus the coefficients of the P_j are the columns of V_n^{-1}. Let $P(Z) = \prod_{1 \leq l \leq n}(Z - k_l)$, the *master polynomial*, which can be computed in $\mathcal{O}(n^2)$ multiplications. Then we calculate $\hat{P}_j(Z) = \prod_{\substack{l \neq j \\ 1 \leq l \leq n}} (Z - k_l) = P(Z)/(Z - k_j)$. Thus $\hat{P}_j(Z)$ can be computed using polynomial division, and then $P_j(Z) = \hat{P}_j(Z)/\hat{P}_j(k_j)$ can be computed using scalar division. This requires only $\mathcal{O}(n)$ space and time. We want to calculate $\bar{X} = (X_1, \cdots, X_n)$ such that,

$$\begin{aligned} X_1 + k_1 X_2 + k_1^2 X_3 + \cdots + k_1^{n-1} X_n &= w_1 \\ X_1 + k_2 X_2 + k_2^2 X_3 + \cdots + k_2^{n-1} X_n &= w_2 \\ \vdots \qquad\qquad\qquad\qquad\quad &= \vdots \\ X_n + k_n X_2 + k_n^2 X_3 + \cdots + k_n^{n-1} X_n &= w_n \end{aligned} \quad \Leftrightarrow \quad V_n \cdot \bar{X}^T = \begin{pmatrix} w_1 \\ w_2 \\ \vdots \\ w_n \end{pmatrix}.$$

Then, $\bar{X}^T = V_n^{-1} \cdot (w_1, \cdots, w_n)^T$ and we get

$$\begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix} = \begin{pmatrix} w_1 \cdot coef(P_1, Z^0) \\ \vdots \\ w_1 \cdot coef(P_1, Z^{n-1}) \end{pmatrix} + \cdots + \begin{pmatrix} w_n \cdot coef(P_n, Z^0) \\ \vdots \\ w_n \cdot coef(P_n, Z^{n-1}) \end{pmatrix}. \tag{2.6}$$

Since the inverse of the transpose of a matrix is the transpose of the inverse, this approach can also be applied to a transposed Vandermonde matrix,

which has the following form.

$$\begin{bmatrix} 1 & 1 & \cdots & 1 \\ k_1 & k_2 & \cdots & k_n \\ k_1^2 & k_2^2 & \cdots & k_n^2 \\ \vdots & \vdots & \vdots & \vdots \\ k_1^{n-1} & k_2^{n-1} & \cdots & k_n^{n-1} \end{bmatrix} \tag{2.7}$$

Then, for $V_n^T \cdot \bar{X} = (w_1, \cdots, w_n)^T$, we get $\bar{X} = (V_n^T)^{-1} \cdot (w_1, \cdots, w_n)^T = (V_n^{-1})^T \cdot (w_1, \cdots, w_n)^T$, and thus

$$\begin{pmatrix} X_1 \\ \vdots \\ X_n \end{pmatrix} = \begin{pmatrix} w_1 \cdot coef(P_1, Z^0) \\ \vdots \\ w_1 \cdot coef(P_n, Z^0) \end{pmatrix} + \cdots + \begin{pmatrix} w_n \cdot coef(P_1, Z^{n-1}) \\ \vdots \\ w_n \cdot coef(P_n, Z^{n-1}) \end{pmatrix}. \tag{2.8}$$

We can use this technique to determine $G = gcd(A, B) = c_1 \bar{X}^{\bar{e}_1} + \cdots + c_t \bar{X}^{\bar{e}_t}$, where $\bar{X} = [x_1, \cdots, x_n]$ and $\bar{e}_i$ is a vector representing the degrees. First, we choose a random n-tuple $\bar{\alpha} = (\alpha_1, \cdots, \alpha_n)$. Second, evaluate $G(\bar{\alpha}) = c_1 \bar{\alpha}^{\bar{e}_1} + \cdots + c_t \bar{\alpha}^{\bar{e}_t}$ and denote the value of each monomial $\bar{X}^{\bar{e}_i}$ by m_i, so that $G(\bar{\alpha}) = c_1 m_1 + \cdots + c_t m_t$. Third, we observe that $(\bar{\alpha}^j)^{\bar{e}_i} = (\bar{\alpha}^{\bar{e}_i})^j = m_i^j$. Thus we have the following system of equations in the form of a transposed Vandermonde system.

$$\begin{aligned} G(\bar{\alpha}^0) &= c_1 + c_2 + \cdots + c_t \\ G(\bar{\alpha}^1) &= c_1 m_1 + c_2 m_2 + \cdots + c_t m_t \\ G(\bar{\alpha}^2) &= c_1 m_1^2 + c_2 m_2^2 + \cdots + c_t m_t^2 \\ \vdots & \qquad \vdots \\ G(\bar{\alpha}^{t-1}) &= c_1 m_1^{t-1} + c_2 m_2^{t-1} + \cdots + c_t m_t^{t-1} \ . \end{aligned}$$

We check if all m_i's are distinct. If they are, the above system has a unique solution and can be solved in $\mathcal{O}(t^2)$ time and $\mathcal{O}(t)$ space by calculating the *master polynomial* $P(Z)$ and each $P_j(Z)$, and then using (2.8). This technique is also used in M. Javadi's algorithm for solving non-monic GCD problems.

2.5 Algorithms for Non-monic GCD

Zippel's sparse interpolation works fine for GCD problems monic in x_1, the main variable, or when the leading coefficient of the GCD in the main variable has only one term. If $LC_{x_1}(G)$ has two or more terms, we need to deal with the *normalization problem*. In [6], de Kleine, Monagan and Wittkopf presented the first solution which treats scaling factors as unknown coefficients to be solved for. For example, let $G_f = (Ay^2 + Bu)x^8 + (Czy + Du^2)x^2 + (Ez + Fu^2 + Gu) \in \mathbb{Z}[u, y, z][x]$. Then the algorithm solves a linear the system which has the following form, where c represents a constant and empty entries are zeros.

$$\begin{bmatrix} c & c & & & & & & 1 & & \\ c & c & & & & & & & 1 & \\ c & c & & & & & & & & 1 \\ & & c & c & & & & c & & \\ & & c & c & & & & & c & \\ & & c & c & & & & & & c \\ & & & & c & c & c & c & & \\ & & & & c & c & c & & c & \\ & & & & c & c & c & & & c \end{bmatrix} \begin{bmatrix} C_1 \\ C_2 \\ C_3 \\ C_4 \\ C_5 \\ C_6 \\ C_7 \\ 1 \\ m_2 \\ m_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{2.9}$$

It solves for the coefficient using Gaussian elimination for each block. The algorithm always makes the scale factor m_1 to be 1, and unknowns m_i for other images. Suppose that $G_f = g_1 x_1^{e_1} + g_2 x_1^{e_1} + \cdots + g_t x_1^{e_t}$, where $g_i \in \mathbb{Z}[x_2, \cdots, x_n]$ and the terms are sorted by the number of terms, n_i, in g_i, i.e., $n_1 \leq n_2 \leq \cdots \leq n_t$. Then the total cost of this first approach is $\mathcal{O}(n_1^3 + \cdots + n_s^3)$.

In general we can scale the images based on any coefficient, instead of unknown coefficients. M. Javadi proposed a second approach to solve the normalization problem [4]. The second approach scales the images based on the coefficient with minimum number of terms. If $n_1 = 1$, then the scaling will be similar to monic case. When $n_1 > 1$, with probability $\geq \frac{1}{2}$, we can

find the leading coefficient by solving only a system of size n_1+n_2-1 terms. After finding the leading coefficient using $\mathcal{O}((n_1+n_2)^3)$ time, we can use Zippel's linear space and quadratic time algorithm to find other constant coefficients using $\mathcal{O}(n_i^2)$ time. However, there may be a common factor among g_1 and g_2. Say $G=(y^2+uy)x^8+(uy+u^2)x^2+(z+u^2+u)$, which has assumed form $G_f=(Ay^2+Buy)x^8+(Czy+Du^2)x^2+(Ez+Fu^2+Gu)$. Then $gcd(g_1,g_2)=gcd(y^2+uy,uy+u^2)=(y+u)$. Then the system to solve A, B, C, and D has no solution no matter how many evaluation points we choose. Therefore, we have to solve for a same system as in (2.9) using $\mathcal{O}(n_1^3+\cdots+n_s^3)$ time. Suppose that we need k coefficients of x_1 to form a system with no unlucky factor. We know $k<t$, because $cont_{x_1}A=cont_{x_1}B=1$. Then Javadi's approach requires $\mathcal{O}((n_1+\cdots+n_k)^3+n_{k+1}^2+\cdots+n_t^2)$ time.

3 The Algorithm

Let $A,B\in\mathbb{F}_q[x_1,x_2,\ldots,x_n]$ be polynomials of total degree d over a finite field $\mathbb{F}_q$ with q elements, and $cont_{x_1}(A)=1$ and $cont_{x_1}(B)=1$. Let G be a $gcd(A,B)$, and $A=G\cdot\bar{A}$ and $B=G\cdot\bar{B}$. Hence, $gcd(\bar{A},\bar{B})=1$. When q is small, Brown's MGCD algorithm cannot be applied if there are not enough evaluation points in $\mathbb{F}_q$ to interpolate. We apply Kaltofen and Monagan's approach [5] and consider $A,B\in\mathbb{F}_q[x_n][x_1,\ldots,x_{n-1}]$. By doing this, we can apply Brown's MGCD algorithm on A and B with coefficient ring $\mathbb{F}_q[x_n]$. Since it is an infinite Euclidean domain, we can choose an irreducible polynomial $p(x_n)$ of $\mathbb{F}_q[x_n]$ of degree d so that $\mathbb{K}=\mathbb{F}_q[x_n]/p(x_n)$ is a finite field with q^d elements where q^d must be large enough to interpolate the other variables $x_2,\cdots,x_{n-1}$. We also require q^d large enough so that the probability of getting an unlucky evaluation point in $\mathbb{K}$ or the probability of a term vanishing is small.

Our algorithm combines Brown's modular GCD algorithm with Zippel's sparse interpolation approach. During the interpolation, we also apply Zippel's linear space and quadratic time algorithm to solve Vandermonde matrices. We use Javadi's approach with a refinement to fix the normalization problem.

3.1 MGCD Algorithm Applied on Finite Field

In the MGCD algorithm of Brown's modular algorithm, we could choose $\deg_{x_n} p > \deg_{x_n} G$ in which case one irreducible polynomial $p \in \mathbb{Z}[x_n]$ would be sufficient. However, it is more efficient to apply the *Chinese Remainder Theorem* (CRT) and choose several $p_i(x_n)$ such that $\sum \deg_{x_n} p_i > \deg_{x_n} G$ and $|\mathbb{K}_i|$ is just big enough for evaluation and interpolation, where $\mathbb{K}_i = \mathbb{F}_q[x_n]/p_i(x_n)$. To avoid the *bad "prime" problem*, we also need to choose $p_i \in \mathbb{F}_q[x_n]$ such that p_i does not divide $LC_X(A)$ and $LC_X(B)$ where $X = [x_1, \cdots, x_{n-1}]$. Then, algorithm PGCD is called repeatly to get $G_i = gcd(A/p_i, B/p_i)$. The homomorphism $\phi_{p_i} : \mathbb{F}_q[x_n][x_1, \cdots, x_{n-1}] \rightarrow \mathbb{K}_i[x_1, \cdots, x_{n-1}]$ restricts the polynomial coefficients to be of lower degree than p_i, and hence stops the growth in the coefficients. Then, we can obtain G by applying the CRT on G_i's with moduli p_i's. We will use $\mathbb{K}_i$ to denote the finite field $\mathbb{F}_q[x_n]/p_i$. The algorithm is shown in Figure 1.

3.2 PGCD Algorithm Applied on Finite Field

Similarly, PGCD uses the evaluation homomorphism to reduce the GCD problem for $A, B \in \mathbb{K}_i[x_1, \cdots, x_{n-1}]$ to a series of problems in $\mathbb{K}_i[x_1, \cdots, x_{n-2}]$, i.e., it reduces $(n-1)$-variate problem to a series of $(n-2)$-variate problems. Let $\alpha_j \in \mathbb{K}_i$ such that $x_{n-1} - \alpha_j$ does not divide $LC_{\hat{X}}(A)$ nor $LC_{\hat{X}}(B)$ where $\hat{X} = [x_1, \cdots, x_{n-2}]$. The evaluation homomorphism $\phi_{\alpha_j} : \mathbb{K}_i[x_1, \ldots, x_{n-1}] \rightarrow \mathbb{K}_i[x_1, \ldots, x_{n-2}]$ maps $A(x_1, \cdots, x_{n-1})$ to $A(x_1, ..., x_{n-2}, \alpha_j) = A_{ij}$ and $B(x_1, \cdots, x_{n-1})$ to $B(x_1, ..., x_{n-2}, \alpha_j) = B_{ij}$. PGCD calls itself recursively and finally reduces the problem to univariate GCD problems which can be solved using *Euclidean algorithm* (EA). To obtain $G_i = gcd(A, B)$, PGCD chooses several α_j's and obtains $G_{ij} = gcd(A_{ij}, B_{ij})$ by recursively calling itself. Then it interpolates the G_{ij}'s and α_j's. The algorithm is shown in Figure 2.

3.3 Sparse Interpolation Applied on Finite Field

When G is sparse, dense interpolation is not efficient. For example, if MGCD chooses irreducible polynomials of degree 5 in $\mathbb{F}_2[z]$ for $G = x^{10} + (y^{10} + z^5 +$

$2z+1)x^2+z^{14} \mod 2$, then it requires three of them polynomials and 11 evaluation values for each of them. Thus, it requires 33 univariate images in total. On the other hand, after we calculate the univariate images using one irreducible polynomial, we obtain the assumed form $G_f = x^{10} + (C_1 y^{10} + C_2)x^2 + C_3$ from a dense interpolation of y. Then we just need two univariate images for each of the other two irreducible polynomials.

Example 2:

Let $G_1 = x^{10} + W_{11}x^2 + W_{12}$ and $G_2 = x^{10} + W_{21}x^2 + W_{22}$ be the images when $y = \alpha_1$ and $y = \alpha_2$ respectively. Then we can set up two linear systems,

$$\begin{bmatrix} \alpha_1^{10} & 1 \\ \alpha_2^{10} & 1 \end{bmatrix} \cdot \begin{bmatrix} C_1 \\ C_2 \end{bmatrix} = \begin{bmatrix} W_{11} \\ W_{21} \end{bmatrix} \qquad \textit{and} \qquad \begin{bmatrix} C_3 \\ C_3 \end{bmatrix} = \begin{bmatrix} W_{12} \\ W_{22} \end{bmatrix}, \tag{3.1}$$

and solve for C_1, C_2, and C_3. Then, it requires only 11 (for the dense interpolation) $+4$ (for two sparse interpolations) $= 15$ univariate images in total. However, if we choose $p_1(z) = z^5 + z + 1$, then the coefficient of x^2 is just $C_1 y^{10}$. Then constant term C_2 is absent in the assumed form, and thus the first system in 3.1 becomes

$$\begin{bmatrix} \alpha_1^{10} \\ \alpha_2^{10} \end{bmatrix} \cdot \begin{bmatrix} C_3 \\ C_3 \end{bmatrix} = \begin{bmatrix} W_{11} \\ W_{21} \end{bmatrix}$$

which may have no solution. Then, the algorithm must restart with another irreducible polynomial. This problem is called the *term vanishing problem.* We estimate the probability of a term vanishing in Section 4. The MGCD and PGCD algorithms with sparse interpolation are shown in Figure 3 and Figure 4, respectively. The sparse interpolation algorithm is shown in 5.

3.4 Quadratic Time Algorithm for Non-monic GCD

M. Javadi's solution to the normalization problem using Zippel's linear space and quadratic time algorithm can be also modified to work for non-monic GCD problems over finite fields with small cardinality. For example, $G = (y^2+1)x^{10} + (y^{10}+z)x^2 + (y^2+y+z^3+2) \in \mathbb{F}_3[z]/(z^5+2z+1)[x,y]$ has an assumed form $G = (C_1y^2 + C_2)x^{10} + (C_3y^{10} + C_4)x^2 + (C_5y^2 + C_6y + C_7)$.

But it cannot be solved using Zippel's sparse interpolation, because the $LC_X(G) = y^2 + 1 \neq 1$ whereas all the univariate image G_i's in $\mathbb{K}[x]$ are monic. Then, setting $C_1\alpha_i^2 + C_2 = 1$, we would incorrectly assume $C_1 = 0$. Using Javadi's approach, if we let $C_1 = 1$, then we need 3 univariate images to set up a system for determining C_2, C_3, and C_4. Since we want to build Vandermonde matrices, we pick $\alpha \in \mathbb{F}_3[z]/(z^5+2z+1)$ and let $y = 1$, α, α^2. Assume that $G_1 = x^{10} + W_{11}x^2 + W_{12}$, $G_2 = x^{10} + W_{21}x^2 + W_{22}$, $G_3 = x^{10} + W_{31}x^2 + W_{32}$ are the corresponding images. Then we can set up the systems as follows.

$$\begin{aligned} C_3 + C_4 &= (1 + C_2)W_{11} \\ C_3\alpha^{10} + C_4 &= (\alpha + C_2)W_{21} \\ C_3\alpha^{20} + C_4 &= (\alpha^2 + C_2)W_{31} \end{aligned} \quad \Leftrightarrow \quad \begin{bmatrix} 1 & 1 & -W_{11} \\ \alpha^{10} & 1 & -W_{21} \\ \alpha^{20} & 1 & -W_{31} \end{bmatrix} \cdot \begin{bmatrix} C_3 \\ C_4 \\ C_2 \end{bmatrix} = \begin{bmatrix} W_{11} \\ \alpha W_{21} \\ \alpha^2 W_{31} \end{bmatrix}$$

This requires $\mathcal{O}(t^3)$ time to solve the inverse of the square matrix by Gaussian elimination, where t is the dimension of the matrix. After C_2 is solved, the system for solving C_5, C_6 and C_7 is the transpose of a Vandermonde matrix, namely

$$\begin{bmatrix} 1 & 1 & 1 \\ \alpha^2 & \alpha & 1 \\ \alpha^4 & \alpha^2 & 1 \end{bmatrix} \cdot \begin{bmatrix} C_5 \\ C_6 \\ C_7 \end{bmatrix} = \begin{bmatrix} (1 + C_2)W_{13} \\ (\alpha + C_2)W_{23} \\ (\alpha^2 + C_2)W_{33} \end{bmatrix}.$$

This requires only $\mathcal{O}(t^2)$ time and $\mathcal{O}(t)$ space using Zippel's algorithm [8]. However, if G is in $\mathbb{F}_2[z]/(z^5 + z + 1)[x, y]$, then choosing $y = 1$ would make $LC_x(G)|_{y=1} = y^2 + 1|_{y=1} \equiv 0 \mod 2$. In other words, $LC_x(A)|_{y=1} = 0$ and $LC_x(B)|_{y=1} = 0$. Hence, $y = 1$ is a *bad evaluation point*. In fact, the probability of $LC_{x_1}(G) \mod I = 0$ where $I =< x_2-1, x_3-1, \cdots, x_{n-1}-1 >$ is relatively high. A restart of the algorithm does not solve the problem because Zippel's LinSpaceSol algorithm always chooses $\alpha = (1, 1, \cdots, 1)$ as the first evaluation point. Therefore, we consider the modified evaluation sequence $y = \alpha$, α^2 $\alpha^3 \in \mathbb{F}_2[z]/(z^5 + z + 1)$ instead. Then, we get the two

systems as follows.

$$\begin{bmatrix} \alpha^{10} & 1 & -W_{11} \\ \alpha^{20} & 1 & -W_{21} \\ \alpha^{30} & 1 & -W_{31} \end{bmatrix} \cdot \begin{bmatrix} C_3 \\ C_4 \\ C_2 \end{bmatrix} = \begin{bmatrix} \alpha W_{11} \\ \alpha^2 W_{21} \\ \alpha^3 W_{31} \end{bmatrix} , \begin{bmatrix} \alpha^2 & \alpha & 1 \\ \alpha^4 & \alpha^2 & 1 \\ \alpha^6 & \alpha^3 & 1 \end{bmatrix} \cdot \begin{bmatrix} C_5 \\ C_6 \\ C_7 \end{bmatrix} = \begin{bmatrix} (\alpha + C_2)W_{13} \\ (\alpha^2 + C_2)W_{23} \\ (\alpha^3 + C_2)W_{33} \end{bmatrix} .$$

Again this requires $\mathcal{O}(t^3)$ time to solve the first system, where t is the dimension of the first square matrix. The second system involves solving the inverse of a transpose of a *generalized* Vandermonde matrix. We consider a general Vandermonde matrix V_t and its inverse V_t^{-1} as follows.

$$V_t \cdot V_t^{-1} = \begin{bmatrix} k_1 & k_1^2 & k_1^3 & \dots & k_1^t \\ k_2 & k_2^2 & k_2^3 & \dots & k_2^t \\ \vdots & \vdots & \vdots & \dots & \vdots \\ k_t & k_t^2 & k_t^3 & \dots & k_t^t \end{bmatrix} \cdot \begin{bmatrix} a_{11} & a_{12} & a_{13} & \dots & a_{1t} \\ a_{21} & a_{22} & a_{23} & \dots & a_{2t} \\ \vdots & \vdots & \vdots & \dots & \vdots \\ a_{t1} & a_{t2} & a_{t3} & \dots & a_{tt} \end{bmatrix} = \mathbf{I}.$$

Consider jth element of the ith row of the product $V_t \cdot V_t^{-1}$ as a polynomial, $P_j(k_i) = a_{1j}k_i + a_{2j}k_i^2 + a_{3j}k_i^3 + \cdot + a_{nj}k_i^t$. Using the similar technique in Section 2, we can let

$$P_j(Z) = \frac{Z}{k_j} \prod_{\substack{l \neq j \\ 1 \leq l \leq t}} \frac{Z - k_l}{k_j - k_l} . \tag{3.2}$$

We can easily verify that $P_j(k_j) = 1$ and $P_j(k_i) = 0 \ \forall i \neq j$. The *master polynomial* for this case is $P(Z) = Z \cdot \prod_{1 \leq l \leq t}(Z - k_l)$. Then we calculate $\hat{P}_j(Z) = Z \cdot \prod_{\substack{l \neq j \\ 1 \leq l \leq t}} (Z - k_l) = P(Z)/(Z - k_j)$. Thus $\hat{P}_j(Z)$ can be computed using polynomial division, and then $P_j(Z) = \dfrac{\hat{P}_j(Z)}{\hat{P}_j(k_j)}$ can be computed using scalar division. Again, this requires only $\mathcal{O}(t)$ space and time to compute each P_j. Then, we can find V_t^{-1} by calculating all P_j's, $1 \leq j \leq t$, and then obtaining the coefficients. To solve $V_t \cdot \bar{C}^T = (w_1, \cdots, w_t)^T$ where $\bar{C} = (C_1, \cdots, C_t)$, we can calculate

$$C_j = w_1 \cdot coef(P_1, Z^j) + \cdots + w_t \cdot coef(P_t, Z^j). \tag{3.3}$$

Since the inverse of the transpose of a matrix is the transpose of the inverse, we can calculate $V_t^T \cdot \bar{C}^T = (w_1, \cdots, w_t)^T$ using the same master polynomial and $P_j(Z)$'s. We have

$$C_j = w_1 \cdot coef(P_j, Z^1) + \cdots + w_t \cdot coef(P_j, Z^t). \tag{3.4}$$

As in Section 2, we write $G = gcd(A, B) = c_1 \bar{X}^{\bar{e}_1} + \cdots + c_t \bar{X}^{\bar{e}_t}$, where $\bar{X} = [x_1, \cdots, x_n]$ and $\bar{e}_i$ is a degree vector. First, we choose a random n-tuple $\bar{\alpha} = (\alpha_1, \cdots, \alpha_n) \in \mathbb{K}^n$. Second, evaluate $G(\bar{\alpha}) = c_1 \bar{\alpha}^{\bar{e}_1} + \cdots + c_t \bar{\alpha}^{\bar{e}_t}$ and denote the value of each monomial by m_i, i.e., $G(\bar{\alpha}) = c_1 m_1 + \cdots + c_t m_t$. Third, we observe that $(\bar{\alpha}^j)^{\bar{e}_i} = (\bar{\alpha}^{\bar{e}_i})^j = m_i^j$. Thus we have the following system of equations in the form of a transposed Vandermonde system.

$$\begin{aligned}
G(\bar{\alpha}^1) &= c_1 m_1 + c_2 m_2 + \cdots + c_t m_t \\
G(\bar{\alpha}^2) &= c_1 m_1^2 + c_2 m_2^2 + \cdots + c_t m_t^2 \\
\vdots \qquad & \qquad \vdots \\
G(\bar{\alpha}^t) &= c_1 m_1^t + c_2 m_2^t + \cdots + c_t m_t^t \, .
\end{aligned}$$

This system can be solved in $\mathcal{O}(t^2)$ time and $\mathcal{O}(t)$ space by calculating the *master polynomial* $P(Z)$ and each $P_j(Z)$, and then calculating each C_i using equation (3.4). This refinement of sparse interpolation is shown in Figure 6.

Figure 1: MGCD: Brown's Modular GCD Algorithm

Input: $A, B \in \mathbb{F}_q[x_n][x_1, \ldots, x_{n-1}]$, nonzero.
Output: G the GCD of A and B.

if $n = 1$ **then**
  $G \leftarrow gcd(A, B)$; # Call univariate GCD algorithm, i.e., Euclidean algorithm
  if $\deg(G) = 0$ **then** $G \leftarrow 1$; **end if**
  Return G;
end if
Let $X = [x_1, \ldots, x_{n-1}]$.
$a \leftarrow cont_X(A)$; $b \leftarrow cont_X(B)$; $A \leftarrow A/a$; $B \leftarrow B/b$; # Remove scalar content
$c \leftarrow gcd(a, b)$; $g \leftarrow gcd(lc_X(A), lc_X(B))$; # univariate GCD
$(M, H, h) \leftarrow (1, 0, 0)$; $l \leftarrow min(\deg_{x_1}(A), \deg_{x_1}(B))$;
$d \leftarrow min(\deg_X(A), \deg_X(B))$; $s \leftarrow \lceil \log_q(2\,d^2) \rceil$;

while true **do**
  Choose an irreducible polynomial $p \in \mathbb{F}_q[x_n]$ such that $p \nmid M$, $p \nmid g$ and $\deg_{x_n}(p) \geq s$.
  $A_p \leftarrow A \mod p$; $B_p \leftarrow B \mod p$; # $A_p, B_p \in \mathbb{F}_{q^s}[x_1, \ldots, x_{n-1}]$
  $g_p \leftarrow g \mod p$; # univariate GCD
  $G_p \leftarrow PGCD(A_p, B_p) \in \mathbb{F}_{q^s}[x_1, \ldots, x_{n-1}] \cup$ FAIL ;
  if $G_p =$ FAIL **then**
    $(M, H, h) \leftarrow (1, 0, 0)$; $l \leftarrow min(\deg_{x_1}(A), \deg_{x_1}(B))$; # restart
  else
    $m \leftarrow \deg_{x_1}(G_p)$;
    $G_p \leftarrow g_p \cdot \text{lcoeff}(G_p)^{-1} \cdot G_p$; # Normalize G_p so that $\text{lcoeff}(G_p) = g_p$
    if $m = 0$ **then**
      Return c;
    else if $m < l$ **then**
      $l \leftarrow m$; $M \leftarrow p$; $h \leftarrow G_p$
    else if $m = l$ **then**
      $H \leftarrow h$;
      Solve $h \equiv H \mod M$ and $h \equiv G_p \mod p$ for $h \in \mathbb{F}_q[x_n][x_1, \cdots, x_n]$ using Chinese Remainder Theorem.
      $M \leftarrow M \cdot p$
    end if
    if $H = h$ **then**
      # Remove content of result and do division check
      $G \leftarrow primpart_X(H)$;
      if $G \mid A$ and $G \mid B$ **then** Return $c \cdot G$ **end if**
    end if
  end if
end while

Figure 2: PGCD: Multivariate GCD Reduction Algorithm

Input: $A, B \in \mathbb{K}[x_1, \ldots, x_k]$ nonzero, where $\mathbb{K} = \mathbb{F}[x_n]/p$ and $p \in \mathbb{F}[x_n]$ irreducible.
Output: G the GCD of A and B.

if $k = 1$ **then**
  $G \leftarrow gcd(A, B)$; # univariate GCD algorithm, i.e., Euclidean algorithm
  if $\deg(G) = 0$ **then** $G \leftarrow 1$; **end if**
  Return G;
end if
Let $X = [x_1, \ldots, x_{k-1}]$.
$a \leftarrow cont_X(A)$; $b \leftarrow cont_X(B)$; $A \leftarrow A/a$; $B \leftarrow B/b$;
$c \leftarrow gcd(a, b)$; $g \leftarrow gcd(lc_X(A), lc_X(B))$; # univariate GCD
$(M, H, h) \leftarrow (1, 0, 0)$; $l \leftarrow min(\deg_{x_1}(A), \deg_{x_1}(B))$;

while true **do**
  Choose a new element $\alpha \in \mathbb{K}$ such that $M(\alpha) \neq 0$ and $g(\alpha) \neq 0$. If there is no such element, then return FAIL. # The coefficient ring is not large enough.
  $A_\alpha \leftarrow A \mod (x_k - \alpha)$; $B_\alpha \leftarrow B \mod (x_k - \alpha)$; $g_\alpha \leftarrow g \mod (x_k - \alpha)$;
  $G_\alpha \leftarrow PGCD(A_\alpha, B_\alpha) \in \mathbb{F}_{q^s}[x_1, \ldots, x_{k-1}] \cup$ FAIL ;
  if $G_\alpha =$ FAIL **then**
    $(M, H, h) \leftarrow (1, 0, 0)$; $l \leftarrow min(\deg_{x_1}(A), \deg_{x_1}(B))$; # restart
  else
    $m \leftarrow \deg_{x_1}(G_\alpha)$;
    $G_\alpha \leftarrow g_\alpha \cdot \text{lcoeff}(G_\alpha)^{-1} \cdot G_\alpha$; # Normalize G_α so that $\text{lcoeff}(G_\alpha) = g_\alpha$
    if $h = 0$ or $m < l$ **then**
      $l \leftarrow m$; $H \leftarrow h$; $h \leftarrow G_\alpha$; $M \leftarrow (x_k - \alpha)$;
    else if $m = l$ **then**
      $H \leftarrow h$;
      Solve $h \equiv H \mod M$ and $h \equiv G_\alpha \mod (x_k - \alpha)$ for $h \in \mathbb{F}_q[x_n][x_1, \cdots, x_k]$ using the Chinese Remainder Theorem.
      $M \leftarrow M \cdot p$
    end if
    if $H = h$ **then**
      # Remove content of result and do division check
      $G \leftarrow primpart_X(H)$;
      if $G \mid A$ and $G \mid B$ **then** Return $c \cdot G$ **end if**
    end if
  end if
end while

Figure 3: SMGCD: MGCD Algorithm with Sparse Interpolation

Input: $A, B \in \mathbb{F}_q[x_n][x_1, \ldots, x_{n-1}]$, nonzero.
Output: G the GCD of A and B

if $n = 1$ **then**
  $G \leftarrow gcd(A, B)$; # Call univariate GCD algorithm, i.e., Euclidean algorithm
  if $\deg(G) = 0$ **then** $G \leftarrow 1$; **end if**
  Return G;
end if
Let $X = [x_1, \ldots, x_{n-1}]$.
$a \leftarrow cont_X(A)$; $b \leftarrow cont_X(B)$; $A \leftarrow A/a$; $B \leftarrow B/b$; # Remove scalar content
$c \leftarrow gcd(a, b)$; $g \leftarrow gcd(lc_X(A), lc_X(B))$; # univariate GCD
$(M, H, h) \leftarrow (1, 0, 0)$;
$d \leftarrow min(\deg_X(A), \deg_X(B))$; $s \leftarrow \lceil \log_q(2\,d^2) \rceil$

while true **do**
  Choose a new irreducible polynomial $p \in \mathbb{F}[x_n]$ such that $p \nmid M$, $p \nmid g$ and $\deg_{x_n}(p) \geq s$.
  $A_p \leftarrow A \mod p$; $B_p \leftarrow B \mod p$; $g_p \leftarrow g \mod p$;
  $G_p \leftarrow PGCD(A_p, B_p)$;
  if $G_p =$ FAIL **then**
    $(M, H, h) \leftarrow (1, 0, 0)$; # restart
  else
    $m \leftarrow \deg(G_p)$;
    $G_p \leftarrow g_p \cdot \text{lcoeff}(G_p)^{-1} \cdot G_p$; # Normalize G_p so that $\text{lcoeff}(G_p) = g_p$
    if $m = 0$ **then** Return c; **end if**
    $G_f \leftarrow G_p$; $M \leftarrow M \cdot p$; $h \leftarrow G_p$;
    while true **do**
      $H \leftarrow h$;
      Choose a new irreducible polynomial $p \in \mathbb{F}[x_n]$ such that $p \nmid M$, $p \nmid g$ and $\deg_{x_n}(p) \geq s$.
      $G_p \leftarrow SGCD(A_p, B_p, G_f)$; $m \leftarrow \deg_{x_1}(G_p)$;
      if $G_p =$ FAIL **then** break; **end if** # Wrong assumed form
      $G_p \leftarrow g_p \cdot \text{lcoeff}(G_p)^{-1} \cdot G_p$;
      Solve $h \equiv H \mod M$ and $h \equiv G_p \mod p$ for $h \in \mathbb{F}_q[x_n][x_1, \cdots, x_n]$ using Chinese Remainder Theorem.
      $M \leftarrow M \cdot p$;
      if $h = H$ **then**
        $G \leftarrow primpart_X(H)$;
        if $G \mid A$ and $G \mid B$ **then** Return $c \cdot G$ **end if**
      end if
    end while
  end if
end while

Figure 4: SPGCD: PGCD Algorithm with Sparse Interpolation

Input: $A, B \in \mathbb{K}[x_1, \ldots, x_k]$ nonzero, where $\mathbb{K} = \mathbb{F}[x_n]/p$ and $p \in \mathbb{F}[x_n]$ irreducible.
Output: G the GCD of A and B.

if $k = 1$ **then**
  $G \leftarrow gcd(A, B)$; # univariate GCD algorithm, i.e., Euclidean algorithm
  if $\deg(G) = 0$ **then** $G \leftarrow 1$; **end if**
  Return G;
end if
Let $X = [x_1, \ldots, x_{k-1}]$.
$a \leftarrow cont_X(A)$; $b \leftarrow cont_X(B)$; $A \leftarrow A/a$; $B \leftarrow B/b$;
$c \leftarrow gcd(a, b)$; $g \leftarrow gcd(lc_X(A), lc_X(B))$; # univariate GCD
$(M, H, h) \leftarrow (1, 0, 0)$;

while true **do**
  Choose a new element $\alpha \in \mathbb{K}$ such that $M(\alpha) \neq 0$ and $g(\alpha) \neq 0$. If there is no such element, return FAIL.
  $A_\alpha \leftarrow A \mod (x_k - \alpha)$; $B_\alpha \leftarrow B \mod (x_k - \alpha)$; $g_\alpha \leftarrow g \mod (x_k - \alpha)$;
  $G_\alpha \leftarrow PGCD(A_\alpha, B_\alpha)$;
  if $G_\alpha =$ FAIL **then**
    $(M, H, h) \leftarrow (1, 0, 0)$; # restart
  else
    $m \leftarrow \deg(G_\alpha)$;
    $G_\alpha \leftarrow g_\alpha \cdot \text{lcoeff}(G_\alpha)^{-1} \cdot G_\alpha$; # Normalize G_α so that lcoeff$(G_\alpha) = g_\alpha$
    if $m = 0$ **then** Return c **end if**
    $G_f \leftarrow G_\alpha$; $M \leftarrow M \cdot (x_k - \alpha)$; $h \leftarrow G_\alpha$;
    while true **do**
      $H \leftarrow h$;
      Choose a new element $\alpha \in \mathbb{K}$ such that $M(\alpha) \neq 0$ and $g(\alpha) \neq 0$. If there is no such element, return FAIL.
      $G_\alpha \leftarrow SGCD(A_\alpha, B_\alpha, G_f)$;
      if $G_\alpha =$ FAIL **then** break; **end if** # Wrong assumed form
      $G_\alpha \leftarrow g_\alpha \cdot \text{lcoeff}(G_\alpha)^{-1} \cdot G_\alpha$;
      Solve $h \equiv H \mod M$ and $h \equiv G_\alpha \mod (x_k - \alpha)$ for $h \in \mathbb{F}_q[x_n][x_1, \cdots, x_k]$ using the Chinese Remainder Theorem.
      $M \leftarrow M \cdot (x_k - \alpha)$;
      if $H = h$ **then**
        # Remove content of result and do division check
        $G \leftarrow primpart_X(H)$;
        if $G \mid A$ and $G \mid B$ **then** Return $c \cdot G$ **end if**
      end if
    end while
  end if
end while

Figure 5: SparseInterp: Sparse Interpolation for Monic GCD

Input: $A, B \in \mathbb{K}[x_1, \ldots, x_k]$ nonzero, where $\mathbb{K} = \mathbb{F}[x_n]/p$ and $p \in \mathbb{F}[x_n]$ irreducible.
G_f the assumed form.

Output: G the GCD of A and B.

Let $X = [x_1, \ldots, x_{k-1}]$.
$a \leftarrow cont_X(A)$; $b \leftarrow cont_X(B)$; $A \leftarrow A/a$; $B \leftarrow B/b$; # Remove scalar content
$c \leftarrow gcd(a, b)$; $g \leftarrow gcd(lc_X(A), lc_X(B))$; # univariate GCD
Let $\bar{X} = [x_2, \ldots, x_k]$.

Write $G_f = g_1 x_1^{d_1} + g_2 x_1^{d_2} + \cdots + g_t x_1^{d_t}$, where t is the number of terms in G_f written in this form, and $g_i = C_{i1}\bar{X}^{\bar{e}_{i1}} + \cdots + C_{in_i}\bar{X}^{\bar{e}_{in_i}}$ with $\bar{e}_{ij} \in \mathbb{Z}_{\geq 0}^{k-1}$ be degree vectors and n_i be the number of terms in g_i. C_{ij}'s are yet to be determined. Sort terms in G_f by n_i's, i.e., $n_1 \leq n_2 \leq \cdots \leq n_t$.
for i from 1 to $n_t + 1$ **do**
  Let $\bar{\alpha}$ be a new $(k-1)$-tuple of $\mathbb{K}^{k-1}$ such that $LC_{x_1}(A)(\bar{X} = \bar{\alpha}) \neq 0$.
  $A_{\bar{\alpha}} \leftarrow A(\bar{X} = \bar{\alpha})$; $B_{\bar{\alpha}} \leftarrow B(\bar{X} = \bar{\alpha})$;
  Let $G_{\bar{\alpha}} \leftarrow gcd(A_{\bar{\alpha}}, B_{\bar{\alpha}}) \in \mathbb{K}[x_1]$ by the Euclidean algorithm.
  if $\deg(G_{\bar{\alpha}}) \neq \deg_{x_1}(G_f)$ **then** Return FAIL; **end if** # Wrong assumed form
  for j from 0 to $\deg(G_{\bar{\alpha}})$ **do**
    if $\text{coeff}(G_{\bar{\alpha}}, x_1^j) \neq 0$ and $\text{coeff}(G_f, x_1^j) = 0$ **then**
      Return FAIL; # Wrong assumed form
    end if
  end for
  for j from 1 to t **do**
    $M_j \leftarrow M_j$ appends a row of the values of monomials in $g_j(\bar{X} = \bar{\alpha})$;
    $v_j \leftarrow v_j$ appends an entry with value $\text{coeff}(G_j, x_1^j)$;
  end for
end for

$G \leftarrow 0$;
for i from 1 to t **do**
  Solve $M_i C_i = v_i$ for $C_i \in \mathbb{K}^{n_i}$
  if more than one solution **then**
    Return $SparseInterp(A, B, G_f)$; # linearly dependent system
  end if
  if no solution **then** Return FAIL; **end if** # Wrong assumed form
  $g_i \leftarrow C_{i1}\bar{X}^{\bar{e}_{i1}} + \cdots + C_{in_i}\bar{X}^{\bar{e}_{in_i}}$;
  $G \leftarrow g_i \cdot x_1^{d_i} + G$;
end for
Return $c \cdot G$;

Figure 6: LinSpaceInterp: Sparse Interpolation for Non-monic GCD

Input: $A, B \in \mathbb{K}[x_1, \ldots, x_k]$ nonzero, where $\mathbb{K} = \mathbb{F}[x_n]/p$ and $p \in \mathbb{F}[x_n]$ irreducible. G_f the assumed form.

Output: G the GCD of A and B.

Let $X = [x_1, \ldots, x_{k-1}]$, and $\bar{X} = [x_2, \ldots, x_k]$.
$a \leftarrow cont_X(A)$; $b \leftarrow cont_X(B)$; $A \leftarrow A/a$; $B \leftarrow B/b$; # Remove scalar content
$c \leftarrow gcd(a, b)$; $g \leftarrow gcd(lc_X(A), lc_X(B))$; # univariate GCD

Write $G_f = g_1 x_1^{d_1} + g_2 x_1^{d_2} + \cdots + g_t x_1^{d_t}$, where t is the number of terms in G_f written in this form, and $g_i = C_{i1}\bar{X}^{\bar{e}_{i1}} + \cdots + C_{in_i}\bar{X}^{\bar{e}_{in_i}}$ with $\bar{e}_{ij} \in \mathbb{Z}_{\geq 0}^{k-1}$ be degree vectors and n_i be the number of terms in g_i. C_{ij}'s are yet to be determined. Sort terms in G_f by n_i's, i.e., $n_1 \leq n_2 \leq \cdots \leq n_t$.

$l = \max\{n_t, \frac{\sum_{i=1}^{t} n_i - 1}{t-1}\}$; # number of univariate images required

Let $\bar{\alpha}$ be a new $(k-1)$-tuple of $\mathbb{K}^{k-1}$ such that $LC_{x_1}(A)(\bar{X} = \bar{\alpha}) \neq 0$.
for i from 1 to l **do**
  $A_{\bar{\alpha}} \leftarrow A(\bar{X} = \bar{\alpha}^i)$; $B_{\bar{\alpha}} \leftarrow B(\bar{X} = \bar{\alpha}^i)$;
  Let $G_{\bar{\alpha}} \leftarrow gcd(A_{\bar{\alpha}}, B_{\bar{\alpha}}) \in \mathbb{K}[x_1]$ by the Euclidean algorithm.
  if $\deg(G_{\bar{\alpha}}) \neq \deg_{x_1}(G_f)$ **then** Return FAIL; **end if** # Wrong assumed form
  for j from 0 to $\deg(G_{\bar{\alpha}})$ **do**
    if coeff$(G_{\bar{\alpha}}, x_1^j) \neq 0$ and coeff$(G_f, x_1^j) = 0$ **then**
      Return FAIL; # Wrong assumed form
    end if
  end for
  for j from 1 to t **do**
    $M_j \leftarrow M_j$ appends a row of the values of monomials in $g_j(\bar{X} = \bar{\alpha})$;
    $v_j \leftarrow v_j$ appends an entry with value coeff(G_j, x_1^j);
  end for
end for
for j from 1 to t **do**
  Solve $[M_1, \cdots, M_j] \cdot [C_1, \cdots, C_j, 1, m_2, \cdots, m_l]^T = [v_1, \cdots, v_j]$ for $[C_1, \cdots, C_j, 1, m_2, \cdots, m_l]$, i.e., use $g_1, \cdots, g_j$ to find the scaling factors.
  if no solution **then**
    Return FAIL; # wrong assumed form
  else if a unique solution **then**
    Calculate $g_i = C_{i1}\bar{X}^{\bar{e}_{i1}} + \cdots + C_{in_i}\bar{X}^{\bar{e}_{in_i}}$ for all $1 \leq i \leq j$.
    Break; # linearly independent system $\Rightarrow$ done.
  end if
end for
$G \leftarrow 0$;
for i from 1 to t **do**
  if g_i has not been solved **then**
    Solve $M_i C_i = v_i \cdot [1, m_2, \cdots, m_l]^T$ for C_i
    if more than one solution **then** Return $SparseInterp(A, B, G_f)$; **end if**
    if no solution **then** Return FAIL; **end if** # Wrong assumed form
    $g_i \leftarrow C_1\bar{X}^{\bar{e}_{i1}} + \cdots + C_{n_i}\bar{X}^{\bar{e}_{in_i}}$;
  end if
  $G \leftarrow g_i \cdot x_1^{d_i} + G$;
end for
Return $c \cdot G$;

4 Algorithm Analysis

We have seen in Section 2 and Section 3 various problems with the modular and sparse algorithm. In this section, we analyze the probability of success.

4.1 Probability of Getting an Unlucky Evaluation Point

If an unlucky evaluation point is chosen, then the univariate GCD image, which can be identified by its higher degree than other images, is abandoned. This may make the execution time longer, since more images are required. We will prove the following theorem which gives bound on the probability of choosing an unlucky evaluation point.

Theorem 4.1. *Let A, B be polynomials in $\mathbb{F}_q[x_1, ..., x_n]$ where $\mathbb{F}_q$ is a finite field with q elements, and $G = gcd(A, B)$. Let $\bar{A} = A/G, \bar{B} = B/G$, and $d = \max\{\deg A, \deg B\}$. Let T be a bound on the number of terms in G. If $p \in \mathbb{F}_q[x_n]$ is an irreducible polynomial with degree $s \geq \lceil \log_q(2d^2T) \rceil$, then for any $(n-2)$-tuple $\alpha = (r_2, r_3, \ldots, r_{n-1}) \in \mathbb{K}^{n-2}$ where $\mathbb{K} = \mathbb{F}_q[x_n]/p$,*

$$Prob[\deg_{x_1}(gcd(\bar{A}_p(\alpha) \in \mathbb{K}[x_1], \bar{B}_p(\alpha) \in \mathbb{K}[x_1])) > 0] \leq \frac{1}{2\,T}. \tag{4.1}$$

The GCD problem can be approached differently, because there is strong relation between the GCD and the resultant of two polynomials. In the following, R is an arbitrary *unique factorization domain* (UFD).

Definition 4.2. Let $A(x), B(x) \in R[x]$ be nonzero polynomials with $A(x) = \sum_{i=0}^{m} a_i x^i$ and $B(x) = \sum_{i=0}^{l} b_i x^i$. The *Sylvester matrix* of A and B is an $m + l$ by $m + l$ matrix

$$Syl(A, B) = \begin{bmatrix} a_m & a_{m-1} & \ldots & a_1 & a_0 & & \\ & a_m & a_{m-1} & \ldots & a_1 & a_0 & \\ & & \ddots & \ldots & \ldots & \ddots & \\ & & & a_m & \ldots & \ldots & a_0 \\ b_l & b_{l-1} & \ldots & b_1 & b_0 & & \\ & b_l & b_{l-1} & \ldots & b_1 & b_0 & \\ & & \ddots & \ldots & \ldots & \ddots & \\ & & & b_l & \ldots & \ldots & b_0 \end{bmatrix}, \tag{4.2}$$

where the upper part of the matrix consists of n rows of coefficients of $A(x)$ and the lower part consists of m rows of coefficients of $B(x)$. The entries not shown are zero.

Definition 4.3. Let $A(x), B(x) \in R[x]$ be polynomials. The *resultant* of $A(x)$ and $B(x)$, denoted by $Res(A, B)$, is the determinant of the Sylvester matrix of A, B. We also define $Res(0, B) = 0$ for nonzero $B \in R[x]$, and $Res(A, B) = 1$ for nonzero $A, B \in R$. We write $Res_x(A, B)$ if we wish to include the polynomial variable.

Proposition 4.4 (Sylvester's Criterion). [Proposition 8 in Section 3.5 from Cox *et al.* [3]] *Given $A(x), B(x) \in R[x]$ be polynomials of positive degree, the resultant $Res(A, B) \in R$ is an integer polynomial in the coefficients of A and B. Furthermore, A and B have a common factor in $R[x]$ if and only if $Res(A, B) = 0$.*

To prove Theorem 4.1, we still need one more ingredient, the Schwartz-Zippel Theorem. It is a bound for the probability of encountering a root of a polynomial. This helps us estimate the probability of choosing unlucky evaluation point.

Theorem 4.5 (Schwartz-Zippel). *Let $P \in \mathbb{F}[x_1, x_2, \ldots, x_n]$ be a non-zero polynomial of total degree $d \geq 0$ over a field $\mathbb{F}$. Let $\mathcal{S}$ be a finite subset of $\mathbb{F}$ and let $r_1, r_2, \ldots, r_n$ be selected randomly from $\mathcal{S}$, then*

$$Prob[P(r_1, r_2, \ldots, r_n) = 0] \leq \frac{d}{|\mathcal{S}|}. \tag{4.3}$$

In our algorithm, evaluation points are randomly chosen from the finite field $\mathbb{K} = \mathbb{F}_q[x_n]/p(x_n)$ where $\deg_{x_n} p = s$, i.e., $\mathcal{S} = \mathbb{K}$. Since $|\mathbb{K}| = q^s$, equation (4.3) can be rewritten as

$$Prob[P(r_1, r_2, \ldots, r_n) = 0] \leq \frac{d}{q^s}. \tag{4.4}$$

Remark 4.6. The Schwartz-Zippel Theorem is a tight bound on the probability of getting a zero evaluation point. Consider the following example,

$$\begin{aligned} &Over\ \mathbb{F}_3: \\ &\qquad P = x^2 + yx + (y^2 + 2) \\ &y = 0: \qquad P = x^2 + 2 = (x-1)(x-2) \\ &y = 1: \qquad P = x^2 + x + 0 = x(x+1) \\ &y = 2: \qquad P = x^2 + 2x = x(x+2) \end{aligned}$$

We see that $Prob[P = 0] = \dfrac{2}{3} = \dfrac{\deg(P)}{|\ \mathbb{F}_3\ |}$.

Proof of Theorem 4.1: By definition, an evaluation $\alpha = (r_2, r_3, \ldots, r_{n-1})$ is *unlucky* if $\deg_{x_1}(\bar{A}_p(\alpha), \bar{B}_p(\alpha)) > 0$. By Proposition 4.4, we know that

$$\deg_{x_1}(\bar{A}_p(\alpha), \bar{B}_p(\alpha)) > 0 \quad \Longleftrightarrow \quad Res_{x_1}(\bar{A}_p(\alpha), \bar{B}_p(\alpha)) = 0.$$

Hence, we need to establish the probability that the $Res_{x_1}(\bar{A}_p(\alpha), \bar{B}_p(\alpha)) = 0$. We can rearrange the terms of $\bar{A}_p$ and $\bar{B}_p$, and write them in the following way:

$$\bar{A}_p = a_m(x_2, \ldots, x_{n-1}) \cdot x_1^m + a_{m-1}(x_2, \ldots, x_{n-1}) \cdot x_1^{m-1} + \cdots + a_0(x_2, \ldots, x_{n-1})$$

and

$$\bar{B}_p = b_l(x_2, \ldots, x_{n-1}) \cdot x_1^l + b_{l-1}(x_2, \ldots, x_{n-1}) \cdot x_1^{l-1} + \cdots + b_0(x_2, \ldots, x_{n-1})$$

where $a_i, b_j \in \mathbb{K}[x_2, \ldots, x_{n-1}]$, $1 \le i \le m$, $1 \le j \le l$. The total degree of each a_i or b_j is at most d, since the maximum of the total degrees of $\bar{A}$ and $\bar{B}$ are d. The Sylvester Matrix of $\bar{A}_p$ and $\bar{B}_p$ in x_1 is

$$Syl(\bar{A}_p, \bar{B}_p) = \begin{bmatrix} a_m & a_{m-1} & \dots & a_1 & a_0 & & \\ & a_m & a_{m-1} & \dots & a_1 & a_0 & \\ & & \ddots & \dots & \dots & \ddots & \\ & & & a_m & \dots & \dots & a_0 \\ b_l & b_{l-1} & \dots & b_1 & b_0 & & \\ & b_l & b_{l-1} & \dots & b_1 & b_0 & \\ & & \ddots & \dots & \dots & \ddots & \\ & & & b_l & \dots & \dots & b_0 \end{bmatrix}.$$

There are $m + l$ rows in $Syl(\bar{A}_p, \bar{B}_p)$ and each entry is at most degree d, so total degree of the determinant $r \in R[x_2, \dots, x_{n-1}]$ of $Syl(\bar{A}_p, \bar{B}_p)$ is,

$$\deg(r(x_2, \dots, x_{n-1})) \leq (m + l) \cdot d \leq 2d \cdot d = 2d^2. \tag{4.5}$$

However, the Bézout bound provides a better asymptotic bound, namely

$$\deg(r(x_2, \dots, x_{n-1})) \leq \deg(\bar{A}_p) \cdot \deg(\bar{B}_p) \leq d \cdot d = d^2. \tag{4.6}$$

In some cases the Bézout bound may be greater than $(m + l) \cdot d$, so in practice one calculates both bounds and uses the minimum of the two. By the application of Theorem 4.5 (equation 4.4), we get

$$\begin{aligned} Prob\,[\,\deg_{x_1}(\,gcd(\bar{A}_p(\alpha), \bar{B}_p(\alpha))\,) > 0\,] &= Prob[r(\alpha) = 0] \\ &\leq \frac{d^2}{|\mathbb{F}_q[x_n]/p|} = \frac{d^2}{q^s} \leq \frac{d^2}{q^{\log_q(2d^2T)}} = \frac{d^2}{2d^2T} = \frac{1}{2\,T}, \end{aligned}$$

because the algorithm chooses $s \geq \lceil \log_q 2d^2T \rceil$. $\square$

For each sparse interpolation, the algorithm computes at most T univariate image, where T is a bound on the number of terms in G. Thus, the probability that the algorithm chooses an unlucky evaluation point throughout the whole process is

$$Prob[\textit{choosing an unlucky evaluation point}] \leq T \cdot \frac{1}{2\,T} = \frac{1}{2}.$$

4.2 The Probability of a Term Vanishing

In Section 3.3, we mention that if one or more terms are absent from the assumed form G_f, which is calculated from $gcd(A/p_1, B/p_1)$, then it will lead to a failure and trigger a restart of the algorithm. The following theorem gives a bound on the probability of a failure.

Theorem 4.7. *Let A, B be polynomials in $\mathbb{F}_q[x_n][x_1, \cdots, x_{n-1}]$ with degree $\leq d$, and let $G = gcd(A, B)$. Let T be a bound on the number of terms in G. If $p \in \mathbb{F}_q[x_n]$ is an irreducible polynomial with degree $s > \log_q(2\,(n-2)dT)$, then at any level of sparse interpolation, for the assumed form G_f,*

$$Prob\,[\,\textit{one or more terms vanish in } G_f\,] < \frac{1}{2}. \tag{4.7}$$

Proof: Consider the situation where the sparse interpolation is applied on the variable x_{k+1}, i.e., we have obtained the assumed form of G in the variables $x_1, \ldots, x_k$. Let $\bar{X} = [x_1, \ldots, x_k]$ and $\bar{e_i} \in \mathbb{Z}^k_{\geq 0}$ be the degree vector. Then write

$$G = g_t \cdot \bar{X}^{\bar{e}_t} + g_{t-1} \cdot \bar{X}^{\bar{e}_{t-1}} + \cdots + g_0 \tag{4.8}$$

where $g_i \in \mathbb{F}_q[x_n][x_{k+1}, \ldots, x_{n-1}]$, and the number of terms, t, is yet to determined. We want to establish the probability that one or more terms of G vanish in the assumed form, i.e., the probability that

$$\exists\, g_i(\beta_{k+1}, \ldots, \beta_{n-1}) \;=\; 0 \mod p$$

for a tuple $(\beta_{k+1}, \ldots, \beta_{n-1}) \in \mathbb{K}^{n-k-1}$ chosen at random where $\mathbb{K} = \mathbb{F}_q[x_n]/p$.

Note that $G = gcd(A, B)$, so the total degree of G is at most d. Thus, $\deg(g_i(x_{k+1}, \ldots, x_{n-1})) \leq d$. Since T is a bound on the number of terms in G, then $t \leq T$. Therefore, by Theorem 4.5 (Schwartz Theorem), we have

$$\begin{aligned} & Prob[g_i(r_{k+1}, \ldots, r_{n-1}) = 0] && \leq \frac{d}{q^s} \\ \Longrightarrow \quad & Prob[\,\textit{one or more } g_i \,=\, 0\,] && \leq \sum_{i=1}^{t} \frac{d}{q^s} \;=\; \frac{d\,t}{q^s} \;\leq\; \frac{d\,T}{q^s}. \end{aligned}$$

Now consider the whole process where the sparse interpolation is applied to obtain a GCD using an assumed form G_f. The probability that one or more terms vanish in any of the G_f's throughout the whole process is,

$$\begin{aligned} & Prob\,[\ one\ or\ more\ terms\ vanish\ ] \\ \leq\ & \sum_{k=2}^{n-1} \frac{dT}{q^s} \\ \leq\ & (n-2)\,\frac{dT}{q^s} \\ <\ & \frac{(n-2)dT}{q^{\log_q(2(n-2)dT)}} = \frac{(n-2)dT}{2(n-2)dT} = \frac{1}{2}, \end{aligned}$$

because $s > \log_q \left(2\,(n-2)\binom{d+n-1}{d}\,d\right)$. $\square$

In conclusion for this section, if we choose an irreducible polynomial $p \in \mathbb{F}_q[x_n]$ with degree $s > \max\{\log_q(2d^2T), \log_q(2\,(n-2)dT)\}$, then the probability of failure due to either unlucky evaluation point or term vanishing is at most $\frac{1}{2}$. Since the maximum number of monomials in k variables of degree $\leq d$ is $\binom{d+n-1}{d}$, then $T \leq \binom{d+n-1}{d}$. Thus, we can use $\binom{d+n-1}{d}$ as a bound on the number of terms of G. However, when G is sparse, $T \ll \binom{d+n-1}{d}$. In practice, we choose irreducible polynomial with degree $\geq \log_q(2d^2)$. When the algorithm encounters unlucky evaluation point or term vanishing problems, it will trigger a start of the process.

5 Experiments on Benchmark Examples

We implemented our algorithm and other algorithms in Maple using the RECDEN data structure. The fact that this data structure supports multiple field extensions over $\mathbb{Q}$ and $\mathbb{Z}_p$ allows our implementation to work over finite fields and algebraic number fields. We also execute some examples to compare the time complexity of different approaches.

Table 1: Comparison of Running Time for the Algorithms

	$y^{10}x^{10}$	$y^{20}x^{10}$	$y^{40}x^{10}$	$y^{80}x^{10}$	$y^{10}x^{20}$	$y^{20}x^{20}$	$y^{40}x^{20}$	$y^{80}x^{20}$
Maple	0.828	1.634	4.876	24.75	2.422	5.325	25.79	136.4
MGCD	0.568	1.131	2.874	8.672	0.632	1.318	3.314	10.15
SMGCD	0.515	0.787	1.686	5.074	0.589	0.863	1.923	5.377
LSGCD	0.445	0.630	1.218	3.189	0.475	0.713	1.354	3.431
	$y^{10}x^{40}$	$y^{20}x^{40}$	$y^{40}x^{40}$	$y^{80}x^{40}$	$y^{10}x^{80}$	$y^{20}x^{80}$	$y^{40}x^{80}$	$y^{80}x^{80}$
Maple	10.25	37.31	182.7	886.2	157.7	713.9	>1000	>2000
MGCD	0.804	1.549	3.561	10.33	1.144	1.950	3.951	8.855
SMGCD	0.669	1.005	1.982	5.460	0.903	1.264	2.312	6.091
LSGCD	0.583	0.839	1.497	3.755	0.745	1.021	1.386	3.123

5.1 Sparse Interpolation in Maple

In Section 3.3, we have seen the gain from using Zippel's sparse interpolation algorithm in reducing the number of images required. This gain is significant if the GCD has a large degree but it is sparse. Consider $G = y^{20}x^{20} + zx^{20} + x^5 + 2yx^4 + (2yz + 2z^2 + zu + 1)x^3 + (y^2 + 2yu + zu) \in \mathbb{Z}_3[x, y, z, u]$. The total degree of G is 40, but the number of terms is only 11. We run this example with different degrees of x and y in the leading term $LT(G) = y^{20}x^{20}$, without changing the rest, on a Duo Core CPU computer with 2 GB memory under the Fedora platform. In Table 5.1, we show the different leading terms and the running time in seconds for Maple's GCD function (Maple), Brown's modular GCD algorithm (MGCD), modular GCD algorithm with Zippel's sparse interpolation (SMGCD), and sparse modular GCD algorithm with Zippel's linear space and quadratic time refinement (LSGCD).

We can observe from the data of Table 5.1 that as the degree of y increases the running time of Maple grows rapidly. On the other hand, sparse interpolation is more efficient and stable than Maple's default algorithm. MGCD's running time is proportional to the degree of y, because it requires $\deg_y(G) + 1$ univariate GCD to interpolate y. Since the number of terms in G is unchanged, after the assumed form is obtained the number of univariate images required also remains unchanged for sparse interpolation. Therefore, SMGCD and LSGCD are less sensitive to the increase of the degree of y.

Table 2: Comparion of SMGCD and LSGCD

	$t=4$	$t=5$	$t=6$	$t=7$	$t=8$	$t=9$	$t=10$	$t=11$	$t=12$
SMGCD	3.195	3.829	3.846	6.793	7.488	8.258	8.951	9.228	9.466
LNGCD	2.195	2.585	2.689	2.887	3.302	3.766	3.977	4.107	4.281

5.2 Linear Space and Quadratic Time

Since the example in the previous section has no change in the number of terms, it is not obvious how SMGCD and LSGCD compare to each other. Consider $G := (y^{10} + zy + z + u)x^{20} + (y^{10} + zy + z + u + 1)x^{10} + (y^{10} + zy^2 + z^2 + u^{10})x^5 + (z^{10} + zy)x^3 + (y^2 + y + z)$, which has term counts in the main variable x are $n_1 = 4, n_2 = 4, n_3 = 4, n_4 = 2$, and $n_5 = 3$.

SMGCD and LSGCD use Javadi's solution to solve the normalization problem, so they both choose the coefficients for x^3 and x^0 to form the first system. We will change the number the terms in the coefficients of x^{20}, x^{10}, and x^5, and run SMGCD and LSGCD on it. Table 5.2 shows the running time in seconds when we change $t = n_1 = n_2 = n_3$ from 4 to 12. We can observe that the running time SMGCD is growing faster than LSMGCD with respect to the increase in the number of terms.

6 Conclusion

We have successfully modified Zippel's sparse modular GCD algorithm to work for finite fields with small cardinality. Zippel's sparse interpolation algorithm is more efficient than Brown's algorithm, but may result in a failure that triggers a new call to the algorithm. We analyzed the probability of success. If our algorithm chooses an irreducible polynomial from the coefficient ring with degree large enough, then the probability of unlucky evaluation point and term vanishing problems will be small.

Javadi's solution for normalization problem with Zippel's linear space and quadratic time refinement make use of Vandermonde matrices, since it saves time and space to calculate its inverse. Our algorithm employs a generalized version of the Vandermonde matrix. We introduced a new master polynomial which efficiently calculates the inverse of the matrix.

References

[1] W. S. Brown. On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors. *J. ACM* **18**, 478-504, 1971.

[2] K. O. Geddes, S. R. Czapor, G. Labahn. *Algorithms for Computer Algebra*. p. 300-335, 1992.

[3] D. Cox, J. Little, D. O'Shea. *Ideals, Varieties, and Algorithms*. Third edition. p. 156, 2007.

[4] M. Javadi. A New Solution to the Polynomial GCD Normalization Problem. MOCAA M^3 Workshop, 2008.

[5] E. Kaltofen, M. B. Monagan. On the Genericity of the Modular Polynomial GCD Algorithm. *Proceeding of ISSAC '99*, ACM Press, 59-66, 1999.

[6] J. de Kleine, M. B. Monagan, A. D. Wittkopf. Algorithms for the Non-monic Case of the Sparse Modular GCD Algorithm. *Proceeding of ISSAC '05*, ACM Press, pp. 124-131 , 2005.

[7] R. Zippel. Probabilistic Algorithms for Sparse Polynomials, *P. EUROSAM '79*, Springer-Verlag LNCS, **2**, 216-226, 1979.

[8] R. Zippel. Interpolating Polynomials from their Values. *J. Symbolic Comp.* **9**(3), 375-403, 1990.