Algorithms for the Non-monic Case of the Sparse Modular GCD Algorithm*

Jennifer de Kleine
Department of Mathematics
Simon Fraser University
Burnaby, B.C. Canada.
dekleine@cecm.sfu.ca.

Michael Monagan
Department of Mathematics
Simon Fraser University
Burnaby, B.C. Canada.
mmonagan@cecm.sfu.ca.

Allan Wittkopf
Maplesoft
615 Kumpf Drive
Waterloo, Ont. Canada.
wittkopf@cecm.sfu.ca.

ABSTRACT

Let $G = (4y^2 + 2z)x^2 + (10y^2 + 6z)$ be the greatest common divisor (GCD) of two polynomials $A, B \in \mathbb{Z}[x, y, z]$. Because G is not monic in the main variable x, the sparse modular GCD algorithm of Richard Zippel cannot be applied directly as one is unable to scale univariate images of G in x consistently. We call this the *normalization problem*.

We present two new sparse modular GCD algorithms which solve this problem without requiring any factorizations. The first, a modification of Zippel's algorithm, treats the scaling factors as unknowns to be solved for. This leads to a structured coupled linear system for which an efficient solution is still possible. The second algorithm reconstructs the monic GCD $x^2 + (5y^2 + 3z)/(2y^2 + z)$ from monic univariate images using a sparse, variable at a time, rational function interpolation algorithm.

Categories and Subject Descriptors: I.1.2 [Symbolic and Algebraic Manipulation]: Algorithms – Algebraic algorithms; F.2.1 [Analysis of Algorithms and Problem Complexity]: Numerical algorithms and problems – Computations on Polynomials.

General Terms: Algorithms.

Keywords: Zippel's Algorithm, Polynomial Greatest Common Divisors, Sparse Multivariate Polynomials, Modular Algorithms, Probabilistic Algorithms.

1. INTRODUCTION

Let A, B be polynomials in $\mathbb{Z}[x_1, ..., x_n]$. Let G be their greatest common divisor (GCD) and let $\bar{A} = A/G$, $\bar{B} = B/G$ be their cofactors. Our problem is to compute G, $\bar{A}$ and $\bar{B}$. In [12] (see also [14] for a more accessible reference) Zippel presented a Las Vegas algorithm for computing G when G is monic in the main variable x_1. Zippel's algorithm improves on the running time of Brown's algorithm (see[1]) when G is also sparse. Zippel's algorithm is an *output sensitive* algorithm. Unlike Brown's algorithm, the number of univariate images depends on the size of G, and not on A and B.

Most computer algebra systems use either Zippel's algorithm or Wang's EEZ-GCD algorithm (see [10]) for multivariate GCD computation. Zippel's algorithm is implemented in Macsyma, Magma, and Mathematica. A parallel implementation is described by Rayes, Wang and Weber in [8]. Previous work done to improve the asymptotic efficiency includes that of Zippel in [13], and Kaltofen and Lee in [5].

In this paper we present two new algorithms that extend Zippel's algorithm to the case where G is not monic in the main variable x_1. In section 2 we give a description of Zippel's algorithm and previous approaches made to extend it to the non-monic case. In section 3 we describe our first solution and in section 4 our second solution. We have implemented both algorithms in Maple. In section 5 we make some remarks about their efficiency and implementation.

Although our algorithms do not require any polynomial factorizations, both require that the content of G in the main variable x_1 is 1. The content of G can be computed efficiently by computing the GCD of one coefficient of A, the smallest, with a random linear combination of the other coefficients of A and all coefficients of B in x_1. This requires one recursive GCD computation in $\mathbb{Z}[x_2, ..., x_n]$.

2. ZIPPEL'S ALGORITHM

There are two subroutines in the algorithm. Subroutine M, the main subroutine, computes $G = \text{GCD}(A, B)$ where $A, B \in \mathbb{Z}[x_1, ..., x_n]$. It does this by computing $\text{GCD}(A, B)$ modulo a sequence of primes $p_1, p_2, ...$ then reconstructs G from these images by applying the Chinese Remainder Theorem. The first image G_1 is computed by calling subroutine P with inputs $A \bmod p_1$ and $B \bmod p_1$. Subroutine P, which is recursive, computes $G = \text{GCD}(A, B)$ where A, B in $\mathbb{Z}_p[x_1, ..., x_n]$ for a prime p as follows. If $n = 1$ it uses the Euclidean algorithm. If $n > 1$ it computes $\text{GCD}(A, B)$ at a sequence of random points $\alpha_1, \alpha_2, ... \in \mathbb{Z}_p$ for x_n then reconstructs $G \in \mathbb{Z}_p[x_1, ..., x_n]$ from the images using *dense* interpolation, e.g., Newton interpolation. The first image G_1 is computed by calling subroutine P recursively with inputs $A \bmod \langle x_n - \alpha_1 \rangle$ and $B \bmod \langle x_n - \alpha_1 \rangle$.

In both subroutines, after the first image G_1 is computed, subsequent images are computed using *sparse* interpolations. This involves solving a set of independent linear systems which are constructed based on the form of G_1. The algo-

*Supported by NSERC of Canada and the MITACS NCE of Canada

ISSAC'05, July 24–27, 2005, Beijing, China.

rithm assumes that G_1 is of the correct form, that is, all non-zero terms of the GCD G are present in G_1. This will be true with high probability if the primes are sufficiently large and the evaluation points are chosen at random from $\mathbb{Z}_p$.

We identify three classes of primes and evaluation points which cause problems in the algorithm.

DEFINITION 1. (BAD PRIME AND EVALUATION)
A prime p is bad if $\deg_{x_1}(G \bmod p) < \deg_{x_1}(G)$. *An evaluation point* $(\alpha_1, ..., \alpha_{n-1}) \in \mathbb{Z}_p^{n-1}$ *is bad if* $\deg_{x_1}(G \bmod I) < \deg_{x_1}(G)$ *where* $I = \langle x_2 - \alpha_1, ..., x_n - \alpha_{n-1}\rangle$.

For example, if $A = (3yx+1)(x-y)$ and $B = (3yx+1)(x+y+1)$ then 3 is a bad prime and $y = 0$ is a bad evaluation point. These must be avoided so that the univariate images can be scaled consistently and so that unlucky primes and evaluations can be detected. We may avoid them by choosing p and $(\alpha_1, ..., \alpha_{n-1})$ such that $L(\alpha_1, ..., \alpha_{n-1}) \not\equiv 0 \bmod p$ where $L = \mathrm{lc}_{x_1} A$ is the leading coefficient of A.

DEFINITION 2. (UNLUCKY PRIME AND EVALUATION)
A prime p is unlucky if the cofactors are not relatively prime modulo p, i.e., $\deg_{x_1}(\mathrm{GCD}(\bar{A} \bmod p, \bar{B} \bmod p)) > 0$. *Similarly an evaluation point* $(\alpha_1, ..., \alpha_{n-1}) \in \mathbb{Z}_p^{n-1}$ *is unlucky if* $\deg_{x_1}(\mathrm{GCD}(\bar{A} \bmod I, \bar{B} \bmod I)) > 0$ *where* $I = \langle x_2 - \alpha_1, \ldots, x_n - \alpha_{n-1}\rangle$.

For example, if $\bar{A} = 7x + 6y$ and $\bar{B} = 12x + y$ then $p = 5$ is an unlucky prime and $y = 0$ is an unlucky evaluation point. These must be avoided if G is to be correctly reconstructed. Unlike bad primes and bad evaluation points, they cannot be ruled out in advance. Instead they identify themselves as we encounter a univariate image in x_1 of higher degree than previous univariate images.

DEFINITION 3. (MISSING TERMS)
A prime p is said to introduce missing terms if any integer coefficient of G vanishes modulo p. Similarly, an evaluation $x_n = \alpha$ is said to introduce missing terms if any coefficient in $\mathbb{Z}_p[x_n]$ of G vanishes at $x_n = \alpha$.

For example, if $G = x^2 + 5y^3x + 35 \in \mathbb{Z}[x, y]$, the primes 5 and 7 and the evaluation $y = 0$ cause terms in G to vanish. Zippel's algorithm cannot reconstruct G if it uses them.

EXAMPLE 1. (THE NORMALIZATION PROBLEM)
Consider computing the non-monic bivariate GCD $G = (y+50)x^3 + 100y \in \mathbb{Z}[x, y]$ *from input polynomials* $A = (x - y + 1)\,G$ *and* $B = (x + y + 1)\,G$. *Here G has leading coefficient $y+50$ in the main variable x. Suppose we compute our first bivariate image modulo $p_1 = 13$ and obtain* $G_1 = (y+11)x^3 + 9y \pmod{13}$. *We proceed to compute a second image using sparse interpolation working modulo* 17. *We assume G has the form* $G_f = (y+\alpha)x^3 + \beta y$ *for some* $\alpha, \beta \in \mathbb{Z}_{17}$. *We have at most one unknown per coefficient in x so we evaluate at one random point, $y = 5$, and compute the univariate* GCD $x^3 + 6 \pmod{17}$. *This image is unique up to a scaling factor m. We evaluate G_f at $y = 5$ and equate to obtain:* $(5+\alpha)x^3 + 5\beta = m(x^3+6)$. *The normalization problem is to determine m. In our example, if we knew $L(y) = y+50$, the leading coefficient of G, then m should be* $L(5) = 4 \pmod{17}$ *and we would have* $(5 + \alpha)x^3 + 5\beta = 4x^3 + 7$. *Solving for α and β in $\mathbb{Z}_{17}$, we would obtain* $G_2 = (y + 16)x^3 + 15y$ *the second bivariate image.*

Let $L = \mathsf{lc}_{x_1}(G)$ be the leading coefficient of G. One solution to the normalization problem is to multiply the monic univariate images by the image of a known multiple of L. The solution used in the Macsyma implementation of Zippel's algorithm is to use $\gamma = \mathrm{GCD}(\mathsf{lc}_{x_1}(A), \mathsf{lc}_{x_1}(B))$. Now L divides γ hence $\gamma = \Delta \times L$ for some Δ in $\mathbb{Z}[x_2, ..., x_n]$. If $\Delta = 1$, then this approach works very well. However, if Δ is a non-trivial polynomial then Zippel's algorithm would reconstruct $\Delta \times G$ which might have many more terms than G and it would need to remove Δ from $\Delta \times G$ which would require another GCD computation. A non-trivial example where this will happen is when computing $\mathrm{GCD}(A, A')$, the first GCD computation in a multivariate square-free factorization algorithm.

An ingenious solution is presented by Wang in [9]. Wang determines L by factoring one of the leading coefficients of the input polynomials, A say, then heuristically determining which factors belong to G and which belong to $\bar{A}$. If A and B are sparse, the factorization is usually not hard. Kaltofen in [4] shows how to reduce the factorization to a bivariate factorization and how to make Wang's heuristic work for coefficient rings other than $\mathbb{Z}$. We now present our solutions. Neither requires any factorizations.

3. ALGORITHM LINZIP

In Zippel's algorithm, if *any* coefficient of G with respect to the main variable x_1 is a monomial in $x_2, ..., x_n$, then the normalization is straightforward. Consider the GCD problem from example 1. Notice that the $\mathcal{O}(x^0)$ term in our first GCD image $G_1 = (y + 11)x^3 + 9y$ has a single term coefficient, $9y$. Since we know the exact form, we can scale our univariate GCD images based on this term. Our assumed form becomes $G_f = (\alpha y + \beta)x^3 + (1)y$ for some $\alpha, \beta \in \mathbb{Z}_{17}$. Now we have two unknowns in our $\mathcal{O}(x^3)$ term so we need two evaluation points, neither of which may be 0. We choose $y = 5, 7$, to get the univariate GCDs $x^3 + 6 \pmod{17}$ and $x^3 + 9 \pmod{17}$, respectively. Now we scale the first univariate GCD by $\frac{5}{6} \pmod{17}$, and the second by $\frac{7}{9} \pmod{17}$ before equating, giving $(5\alpha + \beta)x^3 + 5 = 15x^3 + 5$ and $(7\alpha + \beta)x^3 + 7 = 14x^3 + 7$. Solving for α and β in $\mathbb{Z}_{17}$, gives us the bivariate image, $(8y+9)x^3 + y$, which is a scalar multiple of the GCD modulo 17. Thus if (at any level in the recursion) an image has a coefficient in x_1 which is a single term, the normalization problem is easily solved.

The normalization problem essentially reduces to scaling of the univariate GCD images so that the solution of the linear system produces a correct scalar multiple of the GCD. The approach followed now for the general case is quite simple in concept. It is to treat the scaling factors of the computed univariate GCDs as unknowns as well. This results in larger linear systems that may require additional univariate GCD images. We call this the *multiple scaling* case (as opposed to the *single scaling* case).

Scaling of both the univariate GCDs and the coefficients of the assumed form of the multivariate GCD results in a system that is under-determined by exactly 1 unknown (the computation is only determined up to a scaling factor). We fix the scaling factor of the first image to 1. The following example illustrates this approach.

EXAMPLE 2. *Consider the computation of the bivariate* GCD $(3y^2 - 90)x^3 + 12y + 100$. *We obtain* $g \equiv x^3y^2 + 9x^3 + 4y + 3 \pmod{13}$, *and assumed form of the* GCD $g_f = \alpha x^3y^2 + \beta x^3 + \gamma y + \sigma$. *Instead of computing two univariate* GCD *images for the new prime $p_2 = 17$, we compute three,*

choosing $y = 1, 2, 3$ and obtain the GCDs x^3+12, x^3+8, *and* x^3 *respectively. We form the modified system as follows:*

$$\begin{aligned} \alpha x^3 + \beta x^3 + \gamma + \sigma &= m_1\,(x^3 + 12) = x^3 + 12, \\ 4\alpha x^3 + \beta x^3 + 2\gamma + \sigma &= m_2\,(x^3 + 8), \\ 9\alpha x^3 + \beta x^3 + 3\gamma + \sigma &= m_3\,(x^3), \end{aligned}$$

where m_2, m_3 are the new scaling factors, and we have set the first scaling factor m_1 to 1. Solving this system yields $\alpha = 7, \beta = 11, \gamma = 11, \sigma = 1$, with scaling factors $m_2 = 5, m_3 = 6$, so our new GCD *image is given by $g \equiv 7x^3y^2 + 11x^3 + 11y + 1 \pmod{17}$ which is consistent with our* GCD.

We explain why we fix the multiplier m_1 to be 1 instead of fixing a coefficient in the assumed form of the GCD to be 1. In example 2, suppose we set $\alpha = 1$. Suppose we then use the evaluation $y = 0$ which is not bad. Notice that α should be 0 if $y = 0$. Attempting to set $\beta = 1$ will have the same problem for the prime $p = 5$ which is not bad. Fixing a multiplier to be 1 cannot cause an inconsistency because the algorithm avoids bad primes and bad evaluation points.

One might wonder about the efficiency of the multiple scaling case since we are constructing a system that ties together all unknowns through the multipliers. In the single scaling case, each degree in x_1 has an independent subsystem. The trick is to realize that the resulting system is highly structured, and the structure can be exploited to put the solution expense of the multiple scaling case on the same order as the solution expense of the single scaling case.

EXAMPLE 3. *Consider the linear system that must be solved to compute the* GCD *for a problem with the assumed form $g_f = (a_2y^2 + a_1y + a_0)x^2 + (b_2y^3 + b_1y + b_0)x + (c_1y^2 + c_0)$. We require 3 images to have sufficiently many equations to solve for all unknowns. The resulting linear system has 9 equations in the 10 unknowns. It has the structure shown in figure 1 below. The equations are ordered by decreasing degree in x then by image number. The unknowns are in the same order as in the image followed by the scaling factors. The c's denote (possibly) non-zero entries. All entries not shown are zero.*

$$\begin{bmatrix} c & c & c & & & & & 1 & & \\ c & c & c & & & & & & 1 & \\ c & c & c & & & & & & & 1 \\ & & & c & c & c & & c & & \\ & & & c & c & c & & & c & \\ & & & c & c & c & & & & c \\ & & & & & & c & c & c & \\ & & & & & & c & c & & c \\ & & & & & & c & c & & c \end{bmatrix} \begin{bmatrix} a_2 \\ a_1 \\ a_0 \\ b_2 \\ b_1 \\ b_0 \\ c_1 \\ c_0 \\ 1 \\ m_2 \\ m_3 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}$$

Figure 1: Structure for the multiple scaling case.

The solution can be easily computed by solution of a number of smaller subsystems corresponding to the rectangular blocks of non-zero entries augmented with the multiplier columns. Once the subsystems are upper triangular, remaining rows, only involving the multipliers, can be used to compute the multiplier values, which can then be back-substituted into the subsystems to obtain the image coefficients.

This approach solves the normalization problem but it also introduces another difficulty which is illustrated by the following example.

EXAMPLE 4. *Consider the computation of the bivariate* GCD *$(y + 2)x^3 + 12y^2 + 24y$. By a call to algorithm P we obtain our first image, $g_1 = x^3y+2x^3+12y^2+11y \pmod{13}$, and assumed form of the* GCD *, $g_f = \alpha x^3y+\beta x^3+\gamma y^2+\sigma y$.*

We require at least three univariate GCD *images for the new prime $p_2 = 17$. Choosing $y = 1, 2, 3$ we obtain the* GCDs *$x^3 + 12$, $x^3 + 7$, and $x^3 + 2$ respectively, and form the modified system as follows:*

$$\begin{aligned} \alpha x^3 + \beta x^3 + \gamma + \sigma &= x^3 + 12, \\ 2\alpha x^3 + \beta x^3 + 4\gamma + 2\sigma &= m_2(x^3 + 7), \\ 3\alpha x^3 + \beta x^3 + 9\gamma + 3\sigma &= m_3(x^3 + 2), \end{aligned}$$

In attempting to solve this system, we find that it is under-determined, so we add a new evaluation point, $y = 4$, obtaining a GCD *of $x^3 + 14$, and the new equation*

$$4\alpha x^3 + \beta x^3 + 16\gamma + 4\sigma = m_4(x^3 + 14).$$

The new system of equations is still under-determined. In fact, the system remains under-determined if we continue to choose new evaluation points for y. This is the case for any chosen prime and set of evaluations, so the algorithm fails to find the GCD *for this problem.*

What is not necessarily obvious from example 4 is the cause of the failure, which is the presence of a content in the GCD with respect to the main variable x, namely $y + 2$. In example 4 the content in y is absorbed into the multipliers, so we are unable to obtain a solution for the coefficients in our candidate form as only the relative ratio between terms can be computed. Unfortunately, even if g, the GCD of a and b has no content, certain choices of primes and evaluation points can cause an *unlucky content* to appear in the GCD.

DEFINITION 4. (UNLUCKY CONTENT).
Given $g \in \mathbb{Z}[x_1, ...,x_n]$ with $cont_{x_1}(g) = 1$, a prime p is said to introduce an unlucky content *if $cont_{x_1}(g \bmod p) \neq 1$. Similarly for $g \in \mathbb{Z}_p[x_1, ..., x_n]$ with $cont_{x_1}(g) = 1$, an evaluation $x_i = \alpha_i$ is said to introduce an* unlucky content *if $cont_{x_1}(g \bmod \langle x_i - \alpha_i\rangle) \neq 1$.*

Consider, for example, $g = x(y+1)+y+14$. If we choose $p = 13$ then $g \bmod p$ has a content of $y + 1$, while for any other prime no content is present. Since unlucky contents are rare, we design the algorithm so this problem is not detected in advance, but rather through its effect, so that its detection does not become a bottleneck of the algorithm.

We now present the *LINZIP M* algorithm, which computes the GCD in $\mathbb{Z}[x_1, ..., x_n]$ from a number of images in $\mathbb{Z}_p[x_1, ..., x_n]$, and the *LINZIP P* algorithm, which computes the GCD in $\mathbb{Z}_p[x_1, ..., x_n]$ from a number of images in $\mathbb{Z}_p[x_1, ..., x_{n-1}]$.

ALGORITHM 1 (LINZIP M).

Input: $a, b \in \mathbb{Z}[x_1, ..., x_n]$ *with* $\gcd(\mathrm{cont}_{x_1} a,\ \mathrm{cont}_{x_1} b) = 1$ *and degree bounds $d_{\mathbf{x}}$ on the* GCD *in $x_1, ..., x_n$.*

Output: $g = \mathrm{GCD}(a, b) \in \mathbb{Z}[x_1, ..., x_n]$.

1. *Compute the scaling factor:* $\gamma = \mathrm{GCD}(\mathrm{lc}_{x_1,...,x_n}(a), \mathrm{lc}_{x_1,...,x_n}(b)) \in \mathbb{Z}$.
2. *Choose a random prime p such that $\gamma_p = \gamma \bmod p \neq 0$, and set $a_p = a \bmod p$, $b_p = b \bmod p$, then compute from these a modular* GCD *image $g_p \in \mathbb{Z}_p[x_1, ..., x_n]$ with a call to LINZIP P. If the algorithm returns* **Fail**, *repeat, otherwise set $d_{x_1} = \deg_{x_1}(g_p)$ and continue.*

3 *Assume that g_p has no missing terms, and that the prime is not unlucky. We call the assumed form g_f. There are two cases here:*

3.1 *If there exists a coefficient of x_1 in g_f that is a monomial, then we can use single scaling and normalize by setting the integer coefficient of that monomial to 1. Count the largest number of terms in any coefficient of x_1 in g_f, calling this n_x.*

3.2 *If there is no such coefficient, then multiple scaling must be used. Compute the minimum number of images needed to determine g_f with multiple scaling, calling this n_x.*

4 *Set $g_m = \frac{\gamma_p}{\mathrm{lc}_{x_1,\ldots,x_n}(g_p)} \times g_p \bmod p$ and $m = p$.*

5 *Repeat*

5.1 *Choose a new random prime p such that $\gamma_p = \gamma \bmod p \neq 0$, and set $a_p = a \bmod p$, $b_p = b \bmod p$.*

5.2 *Set $S = \emptyset$, $n_i = 0$.*

5.3 *Repeat*

5.3.1 *Choose $\alpha_2, ..., \alpha_n \in \mathbb{Z}_p \setminus \{0\}$ at random such that $\deg_{x_1}(a_p \bmod I) = \deg_{x_1}(a)$, $\deg_{x_1}(b_p \bmod I) = \deg_{x_1}(b)$ where $I = \langle x_2 - \alpha_2, \ldots, x_n - \alpha_n \rangle$. Set $a_1 = a_p \bmod I$, $b_1 = b_p \bmod I$.*

5.3.2 *Compute $g_1 = \mathrm{GCD}(a_1, b_1)$.*

5.3.3 *If $\deg_{x_1}(g_1) < d_{x_1}$ our original image and form g_f and degree bounds were unlucky, so set $d_{x_1} = \deg_{x_1}(g_1)$ and goto 2.*

5.3.4 *If $\deg_{x_1}(g_1) > d_{x_1}$ our current image g_1 is unlucky, so goto 5.3.1, unless the number of failures $> \min(2, n_i)$, in which case assume p is unlucky and goto 5.1.*

5.3.5 *For single scaling, check that the scaling term in the image g_1 is present. If not, the assumed form must be wrong, so goto 2.*

5.3.6 *Add the equations obtained from equating coefficients of g_1 and the evaluation of $g_f \bmod I$ to S, and set $n_i = n_i + 1$.*

Until $n_i \geq n_x$.

5.4 *We may now have a sufficient number of equations in S to solve for all unknowns in $g_f \bmod p$ so attempt this now, calling the result g_p.*

5.5 *If the system is inconsistent our original image is incorrect (missing terms or unlucky), so goto 2.*

5.6 *If the system is under-determined, then record the degrees of freedom, and if this has occurred twice before with the same degrees of freedom then assume that an unlucky content problem was introduced by the current prime p so goto 5.1. Otherwise we need more images so goto 5.3.1.*

5.7 *The system is consistent and determined. Scale the new image. Set $g_p = \frac{\gamma_p}{\mathrm{lc}_{x_1,\ldots,x_n}(g_p)} \times g_p \bmod p$. Apply the Chinese remainder theorem to update g_m by combining the coefficients of $g_p \in \mathbb{Z}_p[x_1, \ldots, x_n]$ with $g_m \in \mathbb{Z}_m[x_1, ..., x_n]$, updating $m = m \times p$.*

Until g_m has stopped changing for one iteration.

7 *Remove integer content from g_m placing the result in g_c. Test if $g_c \mid a$ and $g_c \mid b$. If yes, return g_c. Otherwise we need more primes, so goto 5.1.*

ALGORITHM 2 (LINZIP P).

Input: *$a, b \in \mathbb{Z}_p[x_1, ..., x_n]$, a prime p, and degree bounds $d_{\mathbf{x}}$ on the GCD in $x_1, ..., x_n$.*

Output: *$g = \mathrm{GCD}(a, b) \in \mathbb{Z}_p[x_1, ..., x_n]$ or* **Fail**.

0 *If the GCD of the inputs has content in x_n return* **Fail**.

1 *Compute the scaling factor: $\gamma = \mathrm{GCD}(\mathrm{lc}_{x_1,\ldots,x_{n-1}}(a), \mathrm{lc}_{x_1,\ldots,x_{n-1}}(b)) \in \mathbb{Z}_p[x_n]$.*

2 *Choose $v \in \mathbb{Z}_p \setminus \{0\}$ at random such that $\gamma \bmod \langle x_n - v \rangle \neq 0$. Set $a_v = a \bmod \langle x_n - v \rangle$, $b_v = b \bmod \langle x_n - v \rangle$, then compute $g_v = \mathrm{GCD}(a_v, b_v) \in \mathbb{Z}_p[x_1, ..., x_{n-1}]$ with a recursive call to LINZIP P ($n > 2$) or via the Euclidean algorithm ($n = 2$). If for $n > 2$ the algorithm returns* **Fail** *or for $n = 2$ we have $\deg_{x_1}(g_v) > d_{x_1}$ then return* **Fail**, *otherwise set $d_{x_1} = \deg_{x_1}(g_v)$ and continue.*

3 *Assume that g_v has no missing terms, and that the evaluation is not unlucky. We call the assumed form g_f. There are two cases here:*

3.1 *If there exists a coefficient of x_1 in g_f that is a monomial, then we can use single scaling and normalize by setting the integer coefficient of that monomial to 1. Count the largest number of terms in any coefficient of x_1 in g_f, calling this n_x.*

3.2 *If there is no such coefficient, then multiple scaling must be used. Compute the minimum number of images needed to determine g_f with multiple scaling, calling this n_x.*

4 *Set $g_{seq} = \frac{\gamma(v)}{\mathrm{lc}_{x_1,\ldots,x_{n-1}}(g_v)} \times g_v \bmod p$ and $v_{seq} = v$.*

5 *Repeat*

5.1 *Choose a new random $v \in \mathbb{Z}_p \setminus \{0\}$ such that $\gamma \bmod \langle x_n - v \rangle \neq 0$ and set $a_v = a \bmod \langle x_n - v \rangle$, $b_v = b \bmod \langle x_n - v \rangle$.*

5.2 *Set $S = \emptyset$, $n_i = 0$.*

5.3 *Repeat*

5.3.1 *Choose $\alpha_2, ..., \alpha_{n-1} \in \mathbb{Z}_p \setminus \{0\}$ at random such that $\deg_{x_1}(a_v \bmod I) = \deg_{x_1}(a)$ and $\deg_{x_1}(b_v \bmod I) = \deg_{x_1}(b)$ where $I = \langle x_2 - \alpha_2, ..., x_{n-1} - \alpha_{n-1} \rangle$. Set $a_1 = a_v \bmod I$, $b_1 = b_v \bmod I$.*

5.3.2 *Compute $g_1 = \mathrm{GCD}(a_1, b_1)$.*

5.3.3 *If $\deg_{x_1}(g_1) < d_{x_1}$ then our original image and form g_f and degree bounds were unlucky, so set $d_{x_1} = \deg_{x_1}(g_1)$ and goto 2.*

5.3.4 *If $\deg_{x_1}(g_1) > d_{x_1}$ then our current image g_1 is unlucky, so goto 5.3.1, unless the number of failures $> \min(1, n_i)$, in which case assume $x_n = v$ is unlucky and goto 5.1.*

5.3.5 *For single scaling, check that the scaling term in the image g_1 is present. If not, the assumed form must be wrong, so goto 2.*

5.3.6 *Add the equations obtained from equating coefficients of g_1 and the evaluation of $g_f \bmod I$ to S, and set $n_i = n_i + 1$.*

Until $n_i \geq n_x$.

5.4 *We should now have a sufficient number of equations in* S *to solve for all unknowns in* $g_f \bmod p$ *so attempt this now, calling the result* g_v.

5.5 *If the system is inconsistent our original image is incorrect (missing terms or unlucky), so goto 2.*

5.6 *If the system is under-determined, then record the degrees of freedom, and if this has occurred twice before with the same degrees of freedom then assume the content problem was introduced by the evaluation of* x_n *so goto 5.1. Otherwise we need more images so goto 5.3.1.*

5.7 *The system is consistent and determined. Scale the new image* g_v. *Set* $g_{seq}{=}g_{seq}, \frac{\gamma(v)}{\mathrm{lc}_{x_1,\ldots,x_{n-1}}(g_v)} \times g_v$, $v_{seq} = v_{seq}, v$.

Until we have $d_{x_n} + \deg_{x_n}(\gamma) + 1$ *images.*

6 *Reconstruct our candidate* GCD g_c *using Newton interpolation (dense) on* g_{seq},v_{seq}, *then remove any content in* x_n.

7 *Probabilistic division test: Choose* $\alpha_2, ..., \alpha_n \in \mathbb{Z}_p$ *at random such that for* $I = \langle x_2 - \alpha_2, ..., x_n - \alpha_n \rangle$ *and* $g_1 = g_c \bmod I$ *we have* $\deg_{x_1}(g_1) = \deg_{x_1}(g_c)$. *Then compute* $a_1 = a \bmod I$, $b_1 = b \bmod I$ *and test if* $g_1 \mid a_1$ *and* $g_1 \mid b_1$. *If yes return* g_c, *otherwise goto 2.*

We make some remarks before discussing the correctness and termination of the algorithm.

1. The degree bound of the GCD in the main variable x_1 is used to detect unlucky primes and evaluations, but only those that involve x_1. We update this degree bound *whenever* we compute a GCD of lower degree in x_1. The degree bounds of the GCD in the non-main variables $x_2, ..., x_n$ are used to compute the number of images needed for the interpolation in step 6 of *LINZIP P*. They are not updated by the algorithm. The degree bound for a variable can be obtained by evaluating the inputs mod a random prime and set of evaluations for all but that variable, then as long as the prime and evaluations are not bad, the degree of the univariate GCD is a bound on the degree of the multivariate GCD for that variable.

2. The number of required images for the multiple scaling case computed in step 3.2 can be the same as the number of required images for the single scaling case computed in step 3.1, and no more than 50% higher. The worst case is quite infrequent. It will only occur when there are only two coefficients with respect to the main variable, each having exactly the same number of terms. The extra expense of this step can usually be reduced by an intelligent choice of the main variable x_1. The exact formula for the number of images needed for a problem with coefficients having term counts of $n_1, ..., n_s$ and a maximum term count of $n_{\max}$ is given by $\max(n_{\max}, \lceil (\sum_{i=1}^{s} n_i - 1)/(s-1) \rceil)$. The complexity of *LINZIP* is otherwise the same as that of Zippel's original algorithm. For a detailed asymptotic analysis of the *LINZIP* algorithm, the interested reader may consult [11].

3. The check in step 0 of *LINZIP P* is used to detect an unlucky content in the initial GCD introduced higher up in the recursion by either a prime or evaluation. We note that this approach only requires computation of *univariate* contents to detect the problem as any content in the GCD will eventually show up as a univariate content as we evaluate $x_n, x_{n-1}, ...$

4. The check in step 5.6 of either algorithm is intended to check for an unlucky content introduced by the evaluation (*LINZIP P*) or prime (*LINZIP M*) chosen in step 5.1 of both algorithms. Since it is possible that a new random image from step 5.3.1 does not constrain the form of the GCD (even without the content problem) we check for multiple failures before rejecting the current iteration of loop 5.

5. The *LINZIP P* algorithm performs one probabilistic univariate division test in step 7 instead of testing if $g_c \mid a$ and $g_c \mid b$. This check is substantially less expensive than a multivariate trial division, though there is still a chance that the test fails to detect an incorrect answer, so the termination division test in *LINZIP M* must be retained.

6. Random evaluation points are chosen from $\mathbb{Z}_p \backslash \{0\}$ rather than $\mathbb{Z}_p$ because zero evaluations are likely to cause missing terms in the assumed form, and possibly scaling problems when normalizing images.

To verify the correctness of this algorithm, in addition to the standard issues with modular algorithms we need also verify that the images are scaled consistently to allow the image reconstruction to proceed. We need to consider 4 main problems, namely bad primes or evaluations, unlucky contents, unlucky primes or evaluations, and missing terms in an initial image.

Bad primes and bad evaluations: The treatment of bad primes and bad evaluations is straightforward. It is handled for the first prime or evaluation by the check that γ does not evaluate to 0 in step 2 of the algorithms, handled for subsequent primes or evaluations by the check that γ does not evaluate to 0 in step 5.1 of the algorithms, and handled for the univariate images in step 5.3.1 of the algorithms.

Unlucky content: The unlucky content problem for the first prime or first evaluation is treated in step 0 of *LINZIP P* by the single variable content check. As in point 3 above we emphasize that this check will always detect the problem at some level of the recursion, specifically the level containing the last variable contained in the unlucky content (as all the other variables in the content have been evaluated, so the content becomes univariate). There is no efficient way to detect where such an unlucky content was introduced. It may have been introduced by the prime chosen in *LINZIP M* or any evaluation in prior calls (for x_j with $j > n$) to *LINZIP P* in the recursion. Thus *LINZIP P* fails all the way back up to *LINZIP M* which restarts with a new prime. This strategy is efficient, as only evaluations (modular and variable) and other single variable content checks have been performed before such a failure is detected.

The introduction of an unlucky content by the prime or evaluation chosen in step 5.1 of either algorithm will be handled in the combination of steps 5.4 and 5.6. The result is a system with additional degrees of freedom, so this *always* results in an under-determined system. The check in step 5.6 handles this, as eventually we will obtain a solution for all variables but the free ones resulting from the unlucky

content, so the degrees of freedom will stabilize, and we will go back to step 5.1 choosing a new prime or evaluation.

Unlucky primes and unlucky evaluations: The treatment of unlucky primes and evaluations is less straightforward. First we consider an unlucky evaluation in step 2 of *LINZIP P* for x_n for which the factor added to the GCD depends upon x_1. If the degree bound d_{x_1} is tight, then this will be detected at a lower level of the recursion by step 2 of *LINZIP P* when $n = 2$. If the degree bound d_{x_1} is not tight, then the GCD computed in that step may be unlucky, but we proceed with the computation. Once we reach loop 5, we begin to choose new evaluation points for x_n. With high probability we will choose a new point that is not unlucky in step 5.1, the problem will be detected in step 5.3.3. In the worst case, all evaluations in step 5.1 may also be unlucky, introducing the same factor to the GCD, and we will proceed to step 6, and reconstruct an incorrect result. Note that if the factor is in fact different, then the equations accumulated in step 5.3.5 will most likely be inconsistent, and this problem will most likely be detected in steps 5.4 and 5.5. Step 7 will again perform checks much like those in step 5.3.3, and will detect this problem with high probability, but if it does not, an invalid result may be returned from *LINZIP P*. If we continue to choose unlucky evaluations we will eventually return an incorrect image to *LINZIP M*.

This problem (as well as the unlucky prime case for step 2 of *LINZIP M*) is handled by the structure of *LINZIP M*. Since the steps are essentially the same, the same reasoning follows, and we need the computation to be unlucky through all iterations of loop 5. Since the form of the GCD is incorrect, it is unlikely that g_m will stabilize, and we will continue to loop. If g_m does stabilize, the invalid image will not divide a and b, so step 7 will put us back into the loop. Now within that loop, which cannot terminate until we have found the GCD, step 5.3.4 will eventually detect this problem, as we must eventually find a prime that is not unlucky.

Now consider the case where the unlucky evaluation or prime is chosen in step 2 of either algorithm, and the factor added to the GCD is independent of x_1, that is, it is a content with respect to x_1. This is handled by the same process as the unlucky content problem, specifically it is handled on the way down by step 0 of *LINZIP P*.

Now if an unlucky prime or evaluation occurs in step 5.1 of either algorithm, it must either raise the degree in x_1, in which case it will be detected in step 5.3.4, or it results in an unlucky content. If this content is purely a contribution of the cofactors, then this will not cause a problem for the algorithm, as it reconstructs the new GCD image without that content present (as a result of the assumed form). Hence, the only type of unlucky evaluation that can occur in step 5.3.1 must raise the degree of the GCD in x_1, and thus is handled by step 5.3.4.

Missing terms: If the initial image of g (in either algorithm) has missing terms, the resulting system will likely be inconsistent which will be detected by step 5.5 with high probability. If it is not detected in any iteration of loop 5, then an incorrect image will be reconstructed in step 6 of *LINZIP P*. The additional check in step 7 of *LINZIP P* will detect this problem with the new images with high probability, but if this also fails, then we return an incorrect image from *LINZIP P*. Again assuming a sequence of failures to detect this problem, we arrive at *LINZIP M*. Now we will compute new images in *LINZIP M* until g_c divides both a and b, so the problem must eventually be detected.

Note that the missing term case is the most likely failure case of both algorithms, that is, more likely than unlucky primes, unlucky evaluations, and unlucky contents. The probability of choosing a prime or evaluation that causes a term to vanish is $\mathcal{O}(t/p)$, where t is the number of terms in the polynomial, and p is the prime. Thus the primes need to be much larger than the number of terms.

4. ALGORITHM RATZIP

An alternative way of handling the non-monic case is to use sparse rational function interpolation. The idea is as follows. Suppose we are computing the GCD of two polynomials in $\mathbb{Z}[x, w, y, z]$ with x as the main variable. We will compute the monic GCD in $\mathbb{Z}(w, y, z)[x]$ in the form:

$$x^n + \sum_{i=0}^{n-1} \frac{a_i(w, y, z)}{b_i(w, y, z)} x^i,$$

where $a_i, b_i \in \mathbb{Z}[w, y, z]$, by interpolating the rational function coefficients using a sparse interpolation. For example, if our GCD is $(y + 14)yx^3 + 12y^2x + y + 14$, we compute the monic GCD

$$x^3 + \frac{12y}{y + 14} x + \frac{1}{y}.$$

We then recover the non-monic GCD by multiplying through by the least common multiple of the denominators. In our example, we multiply through by $\text{LCM}(y+14, y) = (y+14)y$ to get our non-monic GCD $(y + 14)yx^3 + 12y^2x + y + 14$.

To illustrate how sparse rational function reconstruction works in general, suppose one of the rational function coefficients is $C = \frac{*w^3+*zy^2}{*z^2+*y^2+wy^3}$, here $*$ indicates an integer. Suppose we have reconstructed C at $w = 5$ to get $C_1 = \frac{*+*zy^2}{*z^2+*y^2+y^3}$. Notice we have normalized the leading coefficient of the denominator to be 1, essentially dividing through by w. We then assume the form to be $C_f = \frac{\alpha(w)+\beta(w)zy^2}{\delta(w)z^2+\gamma(w)y^2+y^3}$, where $\alpha(w), \beta(w), \delta(w), \gamma(w)$ are rational functions in w. We have 4 unknowns so we need 4 equations to solve for the next image, C_2. We do this for as many w values as we need, then perform rational function interpolation in w to obtain $\frac{*w^2+\frac{*}{w}zy^2}{\frac{*}{w}z^2+\frac{*}{w}y^2+y^3}$. Clearing the fractions in w gets us what we want, namely $\frac{*w^3+*zy^2}{*z^2+*y^2+wy^3}$.

EXAMPLE 5. *Consider the computation of the above* GCD $G = (y + 14)yx^3 + 12y^2x + y + 14$ *from input polynomials* $A = (yx + 1)\,G$ *and* $B = (yx + 2)\,G$. *Using* $p_1 = 11$ *we compute our first monic* GCD *image in* $\mathbb{Z}_{11}(y)[x]$ *using dense rational function interpolation. Given a degree bound in* y, $d_y = 2$, *we need* $N = 2d_y + 1 = 5$ *evaluation points to interpolate a rational function of the form* $\frac{ay^2+by+c}{dy^2+ey+f}$ *in* y. *If we do this by constructing a linear system, the rational function interpolation will cost* $\mathcal{O}(N^3)$. *Instead we use the Euclidean Algorithm. We first apply the Chinese Remainder Theorem to reconstruct polynomial coefficients in* y *followed by rational function reconstruction (see [2]). This reduces the cost to* $\mathcal{O}(N^2)$. *We choose* $y = 1, 4, 9, 3, 6$, *to get the* GCD *images in* $\mathbb{Z}_{11}[x]$, $x^3 + 3x + 1$, $x^3 + 10x + 3$, $x^3 + 9x + 5$, $x^3 + 6x + 4$ *and* $x^3 + 8x + 2$, *respectively. We*

interpolate in y to get $x^3 + (6y^4+9y^3+9y^2+10y+2)x + 10y^4+y^3+5y^2+3y+4$ and then apply rational function reconstruction to the coefficients of x to get our first monic GCD *image $G_1 = x^3 + \frac{y}{y+3}x + \frac{1}{y} \in \mathbb{Z}_{11}(y)[x]$, and our assumed form $G_f = x^3 + \frac{\alpha y}{y+\beta}x + \frac{\delta}{y}$.*

Working modulo $p_2 = 13$ we compute a second monic GCD *image in $\mathbb{Z}_{13}(y)[x]$ using sparse rational function interpolation. We have at most two unknowns per coefficient in our main variable x so we need two evaluation points. We evaluate at $y = 1, 6$, and compute the univariate* GCD *images in $\mathbb{Z}_{13}[x]$, $x^3 + 6x + 1$ and $x^3 + x + 11$, respectively. We evaluate G_f at our chosen y values and equate by coefficient to get the following system.*

$$\left.\begin{array}{rclrcl} 6 & = & \frac{\alpha}{1+\beta}, & 1 & = & \frac{\delta}{1} \\ 1 & = & \frac{6\,\alpha}{6+\beta}, & 11 & = & \frac{\delta}{6} \end{array}\right\} \Rightarrow \alpha = 12,\ \beta = 1,\ \delta = 1$$

Substituting back into G_f we get our second monic image in $\mathbb{Z}_{13}(y)[x]$, $G_2 = x^3 + \frac{12\,y}{y+1}\,x + \frac{1}{y}$.

We then apply the Chinese Remainder Theorem to the integer coefficients of the rational functions of G_1 and G_2 to reconstruct our monic GCD *in $\mathbb{Z}(y)[x]$, $x^3 + \frac{12\,y}{y+14}\,x + \frac{1}{y}$. Clearing fractions gives us our non-monic* GCD *in $\mathbb{Z}[x,y]$.*

Algorithm *RATZIP M* computes the GCD in $\mathbb{Z}[x_1,...,x_n]$ from a number of images in $\mathbb{Z}_p(x_2,...,x_n)[x_1]$. After applying the Chinese remainder theorem it must clear the fractions in $\mathbb{Z}(x_2,...,x_n)[x_1]$ which requires further multivariate GCDs which is the disadvantage of this algorithm in comparison with *LINZIP*. Algorithm *RATZIP P* computes the GCD in $\mathbb{Z}_p(x_2,...,x_n)[x_1]$ from a number of images in $\mathbb{Z}_p(x_2,...,x_{n-1})[x_1]$. As for the *LINZIP M* algorithm any content of the GCD with respect to x_1 must be removed before the initial call to the *RATZIP M* algorithm. Unlike in the *LINZIP* algorithms, we do not use single scaling. It is plausible that it may be applied here but it is not straightforward and we have yet to work out the details. For lack of space, only subroutine *RATZIP P* is presented. Since it is similar to *LINZIP P*, the differences are highlighted.

ALGORITHM 3 (RATZIP P).

Input: *$a, b \in \mathbb{Z}_p[x_1,...,x_n]$, a prime p, and degree bounds $d_{\mathbf{x}}$ on the* GCD *in $x_1,...,x_n$.*

Output: *$g = \text{GCD}(a,b) \in$* $\boxed{\mathbb{Z}_p(x_2,...,x_n)[x_1]}$ *or* **Fail**.

0 *If the* GCD *of the inputs has content in x_n return* **Fail**.

1 *Compute the scaling factor:*
$\gamma = \text{GCD}(\text{lc}_{x_1,...,x_{n-1}}(a), \text{lc}_{x_1,...,x_{n-1}}(b)) \in \mathbb{Z}_p[x_n]$.
$\boxed{\textit{If } \gamma = 1 \textit{ then set } RR = False \textit{ else set } RR = True.}$

2 *Choose $v \in \mathbb{Z}_p \setminus \{0\}$ at random such that $\gamma \bmod \langle x_n - v\rangle \neq 0$. Set $a_v = a \bmod \langle x_n - v\rangle$, $b_v = b \bmod \langle x_n - v\rangle$, then compute $g_v = \text{GCD}(a_v, b_v) \in$* $\boxed{\mathbb{Z}_p(x_2,...,x_{n-1})[x_1]}$ *with a recursive call to* $\boxed{RATZIP\ P}$ *($n > 2$) or via the Euclidean algorithm ($n = 2$). If for $n > 2$ the algorithm returns* **Fail** *or for $n = 2$ we have $\deg_{x_1}(g_v) > d_{x_1}$ then return* **Fail**, *otherwise set $d_{x_1} = \deg_{x_1}(g_v)$ and continue.*

3 *Assume g_v has no missing terms, and that the evaluation is not unlucky. We call the assumed form g_f.*

For each coefficient of x_1 in g_f, count the number of terms in the numerator nt and the number of terms in the denominator dt. Take the maximum sum $nt + dt$ over all coefficients and set $n_x = nt + dt - 1$. The -1 is because we normalize the leading coefficients of the denominators to be 1.

4 $\boxed{\textit{Set } g_m = g_v,\ \ m = x_n - v, \textit{ and } N_i = 1.}$

5 *Repeat*

5.1 *Choose a new random $v \in \mathbb{Z}_p \setminus \{0\}$ such that $\gamma \bmod \langle x_n - v\rangle \neq 0$ and set $a_v = a \bmod \langle x_n - v\rangle$, $b_v = b \bmod \langle x_n - v\rangle$.*

5.2 *Set $S = \emptyset$, $n_i = 0$.*

5.3 *Repeat*

5.3.1 *Choose $\alpha_2,...,\alpha_{n-1} \in \mathbb{Z}_p \setminus \{0\}$ at random such that $\deg_{x_1}(a_v \bmod I) = \deg_{x_1}(a)$ and $\deg_{x_1}(b_v \bmod I) = \deg_{x_1}(b)$ where $I = \langle x_2 - \alpha_2,...,x_{n-1} - \alpha_{n-1}\rangle$. Set $a_1 = a_v \bmod I$, $b_1 = b_v \bmod I$.*

5.3.2 *Compute $g_1 = \text{GCD}(a_1, b_1)$.*

5.3.3 *If $\deg_{x_1}(g_1) < d_{x_1}$ then our original image and form g_f and degree bounds were unlucky, so set $d_{x_1} = \deg_{x_1}(g_1)$ and goto 2.*

5.3.4 *If $\deg_{x_1}(g_1) > d_{x_1}$ then our current image g_1 is unlucky, so goto 5.3.1, unless the number of failures $> \min(1, n_i)$, in which case assume $x_n = v$ is unlucky and goto 5.1.*

5.3.5 *Add the equations obtained from equating coefficients of g_1 and the evaluation of $g_f \bmod I$ to S, and set $n_i = n_i + 1$.*

Until $n_i \geq n_x$.

5.4 *We should now have a sufficient number of equations in S to solve for all unknowns in $g_f \bmod p$ so attempt this now, calling the result g_v.*

5.5 *If the system is inconsistent our original image is incorrect (missing terms or unlucky), so goto 2.*

5.6 *If the system is under-determined, then record the degrees of freedom, and if this has occurred twice before with the same degrees of freedom then assume the content problem was introduced by the evaluation of x_n so goto 5.1. Otherwise we need more images so goto 5.3.1.*

5.7 *The system is consistent and determined, so we have a new image g_v.*

Solve $f \equiv g_m \bmod m(x_n)$ and $f \equiv g_v \bmod (x_n - v)$ using the Chinese remainder algorithm for $f \in \mathbb{Z}_p[x_n](x_2,...,x_{n-1})[x_1] \bmod m(x_n) \times (x_n - v)$. Set $g_m = f, m = m(x_n) \times (x_n - v)$, and $N_i = N_i + 1$.

Until $\boxed{N_i \geq d_{x_n} + 1 \textit{ and } (RR = False \textit{ or } N_i \geq 3).}$

6 *Reconstruct*

6.1 *If RR = True then apply rational function reconstruction in x_n and assign the result to g_c. If it fails then we need more points, goto 5.1. For $n > 2$, clear the rational function denominators of $g_c \in \mathbb{Z}_p(x_n)(x_2, ..., x_{n-1})[x_1]$ to obtain $g_c \in \mathbb{Z}_p(x_2, ..., x_n)[x_1]$.*

6.2 *If RR = False then set $g_c = g_m$.*

7 *Probabilistic division test: Choose $\alpha_2, ..., \alpha_n \in \mathbb{Z}_p$ at random such that for $I = \langle x_2 - \alpha_2, ..., x_n - \alpha_n \rangle$ and $g_1 = g_c \bmod I$ we have $\deg_{x_1}(g_1) = \deg_{x_1}(g_c)$. Then compute $a_1 = a \bmod I$, $b_1 = b \bmod I$ and test if $g_1 \mid a_1$ and $g_1 \mid b_1$. If yes return g_c, otherwise goto 2.*

Our implementation of the *RATZIP* algorithm includes the following enhancement. To reconstruct the rational functions in some variable y with degree bound d_y, we need $2d_y + 1$ evaluation points. In fact, we may need fewer points than this, depending on the form of the rational functions being reconstructed. In our implementation we use the *Maximal Quotient Rational Reconstruction* algorithm [7], which uses at most one more evaluation point than the minimum number of points required for the reconstruction to succeed. For example, to reconstruct the rational functions in y of $G = x^3 + \frac{y}{y+3}x + \frac{1}{y}$, we would need 4 points, not 5.

5. IMPLEMENTATION

We have implemented algorithm LINZIP in Maple and have compared it with Maple's default algorithm, an implementation of the EEZ-GCD algorithm of Wang [10]. The linear algebra over $\mathbb{Z}_p$ and univariate polynomial computations over $\mathbb{Z}_p$ and the integer arithmetic are all coded in C. The rest is coded in Maple. Algorithm LINZIP is generally faster when the evaluation points used by the EEZ-GCD algorithm cannot be 0. It is also much less sensitive to unlucky primes and evaluations than the EEZ-GCD algorithm. Otherwise it is generally slower, sometimes more than a factor of 3 slower.

We have also implemented algorithm LINZIP and RATZIP in Maple using the "recden" [3] data structure. This data structure supports multiple field extensions over $\mathbb{Q}$ and $\mathbb{Z}_p$, and hence, will allow us to extend our implementations to work over finite fields and algebraic number fields. The data structure is currently being implemented in the kernel of Maple for improved efficiency. On our data, the two algorithms perform within a factor of 2 of each other. Algorithm LINZIP is generally faster than RATZIP.

A disadvantage of Zippel's algorithm is the large number of univariate images that must be computed for the sparse interpolations which means a large number of evaluations of the inputs. On our test data we find that the percentage of time spent on evaluations was on average 68% and 75% for *LINZIP* and *RATZIP*, respectively. The multivariate trial division in *LINZIP M*(step 7) and *RATZIP M* took 19% and 11% of the time, respectively.

To improve the efficiency of LINZIP and RATZIP, we are implementing the following idea. Instead of evaluating out all but one variable x_1 in LINZIP P and RATZIP P, consider evaluating out all but 2 variables x_1, x_2 and computing the bivariate images using a dense GCD algorithm. Thus think of G as a polynomial in x_1 and x_2 (main variables) with coefficients in $\mathbb{Z}[x_3, ..., x_n]$. If the cost of computing a bivariate image is less than the cost of evaluation mod $I = \langle x_3 - \alpha_3, ..., x_n - \alpha_n \rangle$, overall efficiency is not compromised. If $G \bmod I$ is dense in x_1 and x_2 then we expect a significant reduction in the maximum of the number of terms of the coefficients in x_1 and x_2, hence, a reduction in the maximum size of the linear systems and a reduction in the number of images needed for the sparse interpolations. We also increase the likelihood of not needing to apply the multiple scaling or rational reconstruction methods. Furthermore, we simplify the multivariate GCD computation for the content of G and, in RATZIP M, the final LCM computation.

6. REFERENCES

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

[2] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, UK, 1999.

[3] M. van Hoeij, M. B. Monagan. A Modular GCD Algorithm over Number Fields Presented with Multiple Field Extensions. *P. ISSAC '2002*, ACM Press, 109–116, 2002.

[4] E. Kaltofen. Sparse Hensel lifting. *P. EUROCAL 85*, Springer-Verlag LNCS **2**, 4–17, 1985.

[5] E. Kaltofen, W. Lee. Early Termination in Sparse Interpolation Algorithms. *J. Symbolic Comp.* **36** (3-4), 365–400, 2003.

[6] E. Kaltofen and B. Trager. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *J. Symbolic Comp.* **9**, 301–320, 1990.

[7] M. B. Monagan. Maximal Quotient Rational Reconstruction: An Almost Optimal Algorithm for Rational Reconstruction. *P. ISSAC '2004*, ACM Press, 243–249, 2004.

[8] M. O. Rayes, P. S. Wang, K. Weber. Parallelization of The Sparse Modular GCD Algorithm for Multivariate Polynomials on Shared Memory Multiprocessors, *P. ISSAC '94*, ACM Press, 66–73, 1994.

[9] P. S. Wang. An Improved Multivariate Polynomial Factorization Algorithm. *Math. Comp.* **32** (144), 1215–1231, 1978.

[10] P. S. Wang. The EEZ-GCD algorithm. *ACM SIGSAM Bull.* **14**, 50–60, 1980.

[11] A. D. Wittkopf, Algorithms and Implementations for Differential Elimination, Ph.D. Thesis, Simon Fraser University. (`http://www.cecm.sfu.ca/~wittkopf/WittThesis.pdf`), 2004.

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

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

[14] R. Zippel, *Effective Polynomial Computation*, Kluwer Academic, 1993.