A Sparse Modular GCD Algorithm for Polynomials over Algebraic Function Fields [*]

Seyed Mohammad Mahdi Javadi
School of Computing Science
Simon Fraser University
Burnaby, B.C. Canada.
sjavadi@cecm.sfu.ca.

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

ABSTRACT

We present a first sparse modular algorithm for computing a greatest common divisor of two polynomials $f_1, f_2 \in L[x]$ where L is an algebraic function field in $k \geq 0$ parameters with $r \geq 0$ field extensions. Our algorithm extends the dense algorithm of Monagan and van Hoeij from 2004 to support multiple field extensions and to be efficient when the gcd is sparse. It uses the modified Zippel interpolation of de Kleine, Monagan, and Wittkopf from 2005.

We have implemented our algorithm in Maple. We provide timings demonstrating the efficiency of our algorithm compared to that of Monagan and van Hoeij and with a primitive fraction-free Euclidean algorithm for both dense and sparse gcd problems.

1. INTRODUCTION

Let $F = \mathbb{Q}(t_1, \ldots, t_k)$, $k \geq 0$. For i, $1 \leq i \leq r$, let $m_i(z_1, \ldots, z_i) \in F[z_1, \ldots, z_i]$ be monic and irreducible over $F[z_1, \ldots, z_{i-1}]/\ \langle m_1, \ldots, m_{i-1}\rangle$. Let $L = F[z_1, \ldots, z_r]/\ \langle m_1, \ldots, m_r\rangle$. L is an algebraic function field in k parameters $t_1, \ldots, t_k$ (this also includes number fields). Let f_1 and f_2 be non-zero polynomials in $L[x]$ and let g be their monic gcd. Our problem is, given f_1 and f_2 to compute g or an associate (scalar multiple) of g.

One way to compute g would be to use the Euclidean algorithm. If one does this in $L[x]$ there is an expression swell in F and one must compute with large fractions in F. In [6] Moreno Maza and Rioboo show how to avoid arithmetic with fractions in F for univariate gcd computation modulo a triangular set of polynomials which includes L as a special case. However, the $k+1$ dimensional expression swell that occurs in the coefficients in F makes their algorithm very slow even for inputs of moderate degree.

In [3], Monagan and van Hoeij presented ModGcd, a first modular GCD algorithm for computing gcds over algebraic function fields presented with one field extension. Their algorithm uses a dense interpolation method and hence the time complexity is at least $O(d^k)$ where d bounds the degree of g in the k parameters.

Our algorithm, presented in Section 3, which we call *SparseModGcd* is a sparse modular GCD algorithm. Similar to ModGcd (see [3]), our algorithm uses rational number reconstruction (see [8, 7]) and a variable at a time rational function reconstruction (see [3, 7]) to recover the coefficients of the gcd one parameter at a time. Like ModGcd, our algorithm is also *output sensitive*, i.e., the number of images it computes depends on the size of the gcd g and not on the sizes of f_1 and f_2 which may be much larger.

Unlike ModGcd, our algorithm uses a sparse interpolation to reduce the number of images needed to interpolate g when g is sparse. Zippel's sparse interpolation in [9], was originally developed for gcd computation in $\mathbb{Z}[x_1, \ldots, x_n]$. It is not efficient when the gcd is not monic in the main variable x_1. In [5], Wittkopf, *et al.* presented two techniques to make Zippel's sparse interpolation efficient for non-monic gcds. We adapt one of these techniques for $L[x]$.

Our paper is organized as follows. In Section 2 we present an example of our algorithm showing the main flow of our algorithm. It also shows the reader how Zippel's sparse interpolation works, and illustrates some of the problems that may occur, the use of rational reconstruction and other key design features of the algorithm. We then identify all problems that can occur and provide the theory for the basis of our algorithm which is presented in Section 3.

In Section 4 we compare Maple implementations of our algorithm with ModGcd and with a primitive fraction-free algorithm (see Appendix) on both dense and sparse problem sets to demonstrate its efficiency. Some of our problem sets are multivariate polynomials, that is, problems in $L[x_1, \ldots, x_n]$. In order to use our algorithm for multivariate polynomial inputs we use the same method described by Monagan and van Hoeij in [3], namely, compute the content $c \in L[x_2, \ldots, x_n]$ of the gcd g from the inputs f_1 and f_2, then apply our algorithm to f_1/c and f_2/c treating the polynomial variables $x_2, \ldots, x_n$ as parameters.

We mention also the GCD algorithm of Dahan *et al.* in [2] which computes the gcd of two univariate polynomials modulo a triangular set T of dimension zero over $\mathbb{Q}$, that is, not involving any parameters. Their algorithm, which uses Hensel lifting, is designed to treat also the case where the triangular set does not generate a prime ideal and consequently a zero divisor could be encountered in the Euclidean

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

algorithm. In the conclusion we will describe how one may in principle modify our algorithm to treat zero divisors and explain why we do not use Hensel lifting.

2. AN EXAMPLE

Similar to ModGcd (see [3]), our algorithm works with *primitive associates* of the inputs and the minimal polynomials and computes the primitive associate of the gcd g, which we now define.

Definition 1. *Let $D = \mathbb{Z}[t_1, ..., t_k]$. A non-zero polynomial in $D[z_1, \dots, z_r, x]$ is said to be primitive wrt $(z_1, \dots, z_r, x)$ if the gcd of its coefficients in D is 1. Let f be non-zero in $L[x]$ where L is the algebraic function field previously defined. The denominator of f is the polynomial $den(f) \in D$ of least total degree in $(t_1, \dots, t_k)$ and with smallest integer content such that $den(f)f$ is in $D[z_1, \dots, z_r, x]$. The primitive associate $\check{f}$ of f is the associate of $den(f)f$ which is primitive in $D[z_1, \dots, z_r, x]$ and has positive leading coefficient in a term ordering.*

Example 1. *Let $f = 3tx^2 + 6tx/(t^2 - 1) + 30tz/(1 - t)$ where $m_1(z) = z^2 - t$. Here $f \in L[x]$ where $L = \mathbb{Q}(t)[z]/\langle z^2 - t\rangle$ is an algebraic function field in one parameter t. We have $\operatorname{den}(f) = t^2 - 1$ and $\check{f} = \operatorname{den}(f)f/(3t) = (t^2 - 1)x + 2x - 10z(t + 1)$.*

We demonstrate our algorithm on the following example where note we have used s and t for the parameters instead of t_1 and t_2.

Example 2. *Let $z = \sqrt{1 - st}$ and let*

$$f_1 = (x^2 + \frac{1}{2t}zx + st + 5)(3zx - 3x + 5zt + 10s - 1),$$

$$f_2 = (x^2 + \frac{1}{2t}zx + st + 5)(3zx - 3x + 10s + 4).$$

Here $\check{g} = 2tx^2 + zx + 2st^2 + 10t$. First we compute $\check{f}_1 = 2tf_1$ and $\check{f}_2 = 2tf_2$. To find $\check{g}$ we first compute $g_1 = \gcd(\check{f}_1, \check{f}_2)$ modulo $p_1 = 17$. Let $t = 5$ be our first evaluation point. To compute $g_1(5, s, z, x)$ we need at least two evaluation points to interpolate s but our algorithm will use three since we do not know yet that $\check{g}$ is linear in t. Using the Euclidean algorithm with evaluation points $s = 4, s = 11$ and $s = 3$ we compute

$$h_0 = g_1(5, 4, z, x) = x^2 + 12zx + 8,$$

$$h_1 = g_1(5, 11, z, x) = x^2 + 12zx + 9 \text{ and}$$

$$h_2 = g_1(5, 3, z, x) = x^2 + 12zx + 3.$$

Applying a dense interpolation to $(4, h_0), (11, h_1)$ and $(3, h_2)$ to interpolate s yields

$$G = x^2 + 12zx + 5s + 5.$$

Now we apply rational function reconstruction to the coefficients of G in $\mathbb{Z}_{17}[s]$. We have three points so we attempt to reconstruct rational functions with linear numerators and denominators. We obtain as output $h = G$ and accept it as probably correct since h has the minimum degree in s ($\deg_s(h) = 1$) among all possible rational functions $\frac{A}{B}$ satisfying $\frac{A}{B} \equiv G \bmod (s-4)(s-11)(s-3)$. This method, which is called Maximal Quotient Rational Reconstruction *was first introduced by Monagan in [7]. Since $h|f_1(5, t, z, x)$ and $h|f_2(5, t, z, x)$ we conclude that $g_1(5, s, z, x) = h$. Now in order to compute g_1 we choose another evaluation point $t = 10$. Assuming that $g_1(5, s, z, x)$ is of the correct form, in particular $\deg_s \operatorname{lc}_x(g_1) = 0$,*

$$g_1(10, s, z, x) = 1.x^2 + Azx + Bs + C$$

for some constants A, B and C. If we choose the next evaluation point to be $s = 13$ using the Euclidean algorithm we compute

$$g_1(10, 13, z, x) = x^2 + 6zx + 16.$$

From this we obtain

$$\{A = 6, 13B + C = 16\} \bmod 17.$$

Notice that because of the form of the gcd, we have two independent linear systems, one of dimension 1 and the other of dimension 2 so we need another evaluation point to solve for B and C. Take $s = 14$. Again, using the Euclidean algorithm we obtain

$$g_1(10, 14, z, x) = x^2 + 6zx + 9,$$

hence $14B + C = 9 \bmod 17$. Solving for A, B, C we obtain $A = 6, B = 10$ and $C = 5$ and hence

$$g_1(10, s, z, x) = x^2 + 6zx + 10s + 5.$$

Interpolating the images of g_1 at $t = 5$ and $t = 10$ yields

$$G = x^2 + (9t + 1)xz + st + 5.$$

Applying the rational function reconstruction to the coefficients of G fails. This means we need to choose another evaluation point.

If we choose the next evaluation point $t = 1$, and use $s = 6$, using the Euclidean algorithm we compute

$$g_1(1, 6, z, x) = x^3 + (13z + 4)x^2 + (2z + 1)x + 10z + 13.$$

The degree of this new image wrt x is greater than the degree of the assumed form. In fact no matter, what value of s we choose, this always happens. This is because $t = 1$ is an unlucky *evaluation point which must be detected and rejected.*

Suppose we try $t = 11$. Again using sparse interpolation we obtain

$$g_1(11, s, z, x) = x^2 + 7zx + 11s + 5.$$

By interpolating the images of g_1 at $t = 5$, $t = 10$ and $t = 11$ we get

$$G = x^2 + (10t^2 + 12t + 8)xz + st + 5.$$

This time rational function reconstruction applied to the coefficients of G modulo $(t - 5)(t - 10)(t - 11)$ succeeds and outputs

$$h = x^2 + \frac{9}{t}zx + st + 5.$$

Hence $\check{h} = tx^2 + 9zx + st^2 + 5t$. Since $\check{h}|\check{f}_1 \bmod p_1$ and $\check{h}|\check{f}_2 \bmod p_1$, $g_1 = \check{h}$. Now we apply the integer rational number reconstruction to the coefficients of g_1. It fails because our prime is not big enough, so we need another image. We choose $p_2 = 11$. Let $g_2 = \operatorname{GCD}(\check{f}_1, \check{f}_2) \bmod p_2$. Assuming that g_1 is of the correct form, we know that

$$g_2 = 1.tx^2 + Hzx + (Ist^2 + Jt).$$

Again we use sparse interpolation. We need two images to determine I and J. We choose two random evaluations for $(s,t) \in \mathbb{Z}_p^2$. We obtain $H = 6, I = 1$ and $J = 5$, hence

$$g_2 = tx^2 + 6zx + st^2 + 5t.$$

Now we apply the Chinese Remaindering algorithm to g_1 and g_2 modulo primes $p_1 = 17$ and $p_2 = 11$. This results in

$$G = tx^2 + 94zx + st^2 + 5t \bmod 13 \times 11.$$

This time integer rational number reconstruction succeeds and outputs

$$h = tx^2 + \frac{1}{2}zx + st^2 + 5t,$$

hence

$$\check{h} = 2h = 2tx^2 + zx + 2st^2 + 10t.$$

Since $\check{h}|\check{f}_1$ and $\check{h}|\check{f}_2$, $\check{h} = \mathrm{GCD}(\check{f}_1, \check{f}_2)$ and we are done.

Remark 1: Because we chose evaluation points from $\mathbb{Z}_{17}$ and not the finite ring $\mathbb{Z}_{17}[z]$, the linear system of equations was over $\mathbb{Z}_p$ which means it could not run into zero divisors, which would further complicate the algorithm. Also, because we equate coefficients in $z^i x^j$ instead of x^j, in general, this reduces the number of images needed and the size of the independent linear systems to be solved.

Remark 2: If the leading coefficient of $\check{g}$ in x ($2t$ in our example) was a sum of two or more terms, the interpolation as shown in the example would not work. This is called *Normalization Problem* (or leading coefficient problem). We will solve this problem by adapting one of the solutions introduced by Wittkopf, *et al.* in [5].

2.1 Problems

In the example we encountered an unlucky evaluation point. There are several other problems that may arise depending on the primes and the evaluation points that the algorithm chooses, including the possibility of hitting a zero divisor while using the Euclidean algorithm to compute the univariate images of the gcd. Here we identify all problems. We follow the terminology of van Hoeij and Monagan [3].

Bad primes and bad evaluation points.

Definition 2. *A prime p is said to be a* bad prime *if the leading coefficient of $\check{f}_1$ or $\check{f}_2$ wrt x or any $\check{m}_i$ wrt z_i vanishes mod p. Similarly an evaluation point $t_j = \alpha$ is called a* bad evaluation point *if the degree of $\check{f}_1$ or $\check{f}_2$ wrt x or any $\check{m}_i$ wrt z_i decreases after evaluating at this point.*

Example 3. *Suppose $\check{f}_1 = 28tx^3 + 19ztx + 2t^2 + 10$, $\check{f}_2 = 52zx^2 + 10x + zt^3 - t$ and $m(z) = (s-1)z^2 + 3$. Here $p_1 = 2, p_2 = 7$ and $p_3 = 13$ are bad primes. Also $t = 0$ and $s = 1$ are bad evaluation points.*

The good thing about bad primes and bad evaluation points is that they can be ruled out in advance.

Unlucky primes and unlucky evaluation points.

Definition 3. *A prime p is said to be* unlucky *if $\check{g}_p = \gcd(\check{f}_1, \check{f}_2) \bmod p$ has higher degree in x than the gcd $\check{g}$. Similarly an evaluation point $t_j = \alpha$ is said to be* unlucky *if $\gcd(\check{f}_1(\alpha), \check{f}_2(\alpha)) \bmod p$ has higher degree in x than $\check{g}$.*

Example 4. *Consider the input polynomials*

$\check{f}_1 = (x+z)(x+17t+t^2+z)$ and $\check{f}_2 = (x+t)(x+t^2+z)$.

Here $\check{g} = 1$ but $\check{g}_{17} = \gcd(\check{f}_1, \check{f}_2) \bmod 17 = x+t^2+z$ which obviously has higher degree than $\check{g}$, so $p = 17$ is an unlucky prime. Similarly $t = 0$ is an unlucky evaluation point.

Unlucky primes must be avoided if $\check{g}$ is to be correctly reconstructed. Unlike bad primes, unlucky primes can not be detected and discarded in advance. Brown in [1] showed how to do this in a way that is efficient for $\mathbb{Z}[x]$; whenever images of the gcd do not have the same degree, one keeps only those images of smallest degree and discard the others. The same solution works here. See Theorem 1.

Zero Divisors

Recall that a non-zero element α of a ring R is a zero divisor if there exists a non-zero element $\beta \in R$ s.t. $\alpha\beta = 0$. When we are using the Euclidean algorithm to compute the gcd of $f_1(\alpha_1,\ldots,\alpha_k,x)$, $f_2(\alpha_1,\ldots,\alpha_k,x)$ ($\alpha_1,\ldots,\alpha_k$ are the evaluation points) modulo a prime p, we might encounter a zero divisor, in which case the Euclidean algorithm fails (see [4]). The bigger the prime p is, the smaller the chance of hitting a zero divisor would be.

Example 5. *Let $f_1 = (z+2t)x^2 + tx + z$, $f_2 = zx^3 + tx^2 + (z-2)x + 8t$ and $m(z) = z^2 - t$. Suppose we choose the first prime $p = 7$. If we evaluate the inputs at $t = 2$ we obtain $f_1' = (z-3)x^2 + 2x + z$ and $f_2' = zx^3 + 2x^2 + (z-2)x + 2$. When we run the Euclidean algorithm on the inputs f_1' and f_2' we hit a zero divisor while trying to invert $\mathrm{lc}(f_1') = z - 3$. Note $z^2 - 2 = (z-3)(z+3) \bmod 7$.*

The solution to the problem with zero divisors is described in the next section.

Theorem 1. *Let $f_1, f_2 \in L[x]$ be two non-zero polynomials where $L = F[z_1,\ldots,z_r]/\langle m_1,\ldots,m_r\rangle$ and $F = \mathbb{Q}(t_1,\ldots,t_k)$. Let $\check{g} = \gcd(\check{f}_1, \check{f}_2)$. Let p be a prime and $\alpha = (t_1 = \alpha_1,\ldots,t_k = \alpha_k)$. Suppose that the monic Euclidean algorithm applied to $\check{f}_1(\alpha,x)$ and $\check{f}_2(\alpha,x)$ modulo p does not encounter a zero divisor and outputs g_p (monic in x). If α is not a bad evaluation point and p is not a bad prime, $\deg_x(g_p) \geq \deg_x(\check{g})$. Moreover if $\deg_x(g_p) = \deg_x(\check{g})$ then $g_p = \mathrm{monic}(\check{g}(\alpha,x) \bmod p)$ and we say α is a good evaluation point and p is a good prime.*

Proof: See Monagan and van Hoeij [3]. The difference is the number of field extensions but this has no significant change in the proof in [3].

Theorem 1 and Brown's method for detecting unlucky primes and unlucky evaluation points tells us how to get good primes and good evaluation points which will give us univariate images from which we will reconstruct the multivariate coefficients of $\check{g}$ in $\mathbb{Z}[t_1,...,t_k]$. There are three additional problems to address, the first is due our using sparse interpolation.

Missing Terms

Definition 4. *A prime p is said to introduce* missing terms *if any term of $\check{g}$ vanishes modulo p. Similarly an evaluation point $t_k = \alpha$ is said to introduce* missing terms *if any coefficient of $\check{g}$ in $\mathbb{Z}_p[t_k]$ vanishes at this evaluation point.*

Example 6. *Let $\breve{g} = (t^2 - s + 1)x^3 + 70zx^2 + (13t + 26)$. Here the primes $2, 5, 7$ and 13 introduce missing terms (the first three cause the second term to vanish and the last one makes the last term to vanish). The prime $p = 11$ does not introduce missing terms, but when we are computing the image of the gcd modulo this prime, $t = 9$ makes the last term vanish.*

Unfortunately primes (or evaluation points) which introduce missing terms can not be avoided in advance. We detect an incorrect assumed form g_f as follows. If during the sparse interpolation we compute an image with the same degree as the assumed form g_f in x but with terms in $(x, z_1, \ldots, z_r)$ not present in g_f, the assumed form g_f is wrong. The following example shows what will happen if the assumed form is wrong but the terms are the same.

Example 7. *Suppose $\breve{g} = x + tz + 13z$. Suppose the first prime used is $p_1 = 13$. Then the assumed form for $g(x, z, t)$ will be $g_f = x + Btz$. Now suppose we pick the second prime $p_2 = 11$ and attempt sparse interpolation, equating $\breve{g}(t) = g_f(t)$ for different t. For $t = 1$ we obtain $x+3z = x+Bz \implies B = 3$. For $t = 2$ we obtain $x + 4z = x + 2Bz \implies B = 2$, an inconsistency.*

We argue that an incorrect assumed form g_f will be identified with high probability if we use one more evaluation point than the minimum needed to solve for the unknowns in the assumed form. Once an incorrect assumed form is identified, because we do not know whether it was the current evaluation point or a previous evaluation point or the current prime that caused the missing terms, we must allow our algorithm to restart with a new prime.

Remark 3: Suppose $\breve{g} = (7s^2 + 9t + 1)x^2 + z$. Here $p = 7$ is not bad but we still can not use it because the image of the gcd computed modulo $p = 7$ does not have the correct leading term (see [3]). Note that in our algorithm $p = 7$ causes a missing term and this is treated in the same way we treat missing terms.

Normalization problem

An image of the gcd computed modulo a prime p is unique up to a scaling factor in the integers mod p. This causes a complication in the sparse interpolation when $\mathrm{lc}_x\breve{g}$ has more than one term in the parameters.

Example 8. *Let $z = \sqrt{s+t}$. Suppose the input polynomials are $\breve{f}_1 = (x - s + 1)\breve{g}$* and *$\breve{f}_2 = (x + t + s)\breve{g}$ where*

$$\breve{g} = (15s + t)x^2 + 12s^2xz + 40st$$

is the gcd of $\breve{f}_1$ and $\breve{f}_2$. Suppose we have computed

$$\breve{g}_7 = (s + t)x^2 + 5s^2xz + 5st,$$

our first image modulo $p_1 = 7$. So our assumed form is

$$g_f = (As + Bt)x^2 + Cs^2xz + Dst$$

for some constants A, B, C and D. Now we want to compute the next image of the gcd modulo $p_2 = 11$. Consider the evaluation point $\alpha = (s = 2, t = 1)$, we have

$$\breve{g}_{21} = \gcd(\breve{f}_1(2, 1), \breve{f}_2(2, 1)) = 1 \cdot x^2 + 9xz + 4.$$

The problem is that this image is unique *up to a* scaling factor *m. That is*

$$m(x^2 + 9xz + 4) = (2A + B)x^2 + 4Cxz + 2D, \qquad (*)$$

but we do not know what m is. If we knew the leading coefficient of $\breve{g}$, $\mathrm{lc}_x(\breve{g}) = 15s + t$, then we could easily compute

$$m = \mathrm{lc}_x(\breve{g}(2, 1)) \bmod 11 = 9.$$

Unfortunately there is no easy way of computing $\mathrm{lc}_x\breve{g}$.

Suppose g_f is the assumed form of the gcd and we have computed $h = \gcd(\breve{f}_1(\alpha), \breve{f}_2(\alpha)) \bmod p$. If g is monic then we have the equation $h = g_f(\alpha)$, otherwise we let m be an unknown in (*). Because m is a new unknown, we may need more images to construct a consistent system of linear equations. The above solution was first introduced by Wittkopf, *et al.* in [5]. In their paper, they discuss the efficiency of the multiple scaling case. To see how this affects the structure of the linear systems consider the problem of finding a gcd

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

Figure 1: Structure of the linear system for multiple scaling case

which looks like

$$g_f = (a_1s^2 + a_0)x^2 + (b_1st^2 + b_0)x + (c_1t^2 + c_0)z.$$

Here we need two images. The linear system of equations has a structure shown in Figure 2.1 where all entries not shown are zero and $m_1 = 1$. The solution can be easily computed by solving a number of smaller subsystems corresponding to the rectangular blocks of non-zero entries augmented with the multiplier columns. With this method, the solution expense turns out to be of the same order as the case where $\breve{g}$ is monic.

Unlucky Contents

Definition 5. *For a polynomial $f = a_nx^n + \cdots + a_1x + a_0$ with $a_i \in R$ for $0 \le i \le n$, the* content

$$\mathrm{cont}_x(f) = \gcd(a_0, a_1, \ldots, a_n) \in R.$$

A prime p, is said to introduce an unlucky content *if for two input polynomials f_1, f_2 with gcd $g = \gcd(f_1, f_2)$, $\mathrm{cont}_x(\breve{g}) = 1$ but $\mathrm{cont}_x(\breve{g} \bmod p) \neq 1$. Similarly an evaluation point $t = \alpha$ is said to introduce an unlucky content if $\mathrm{cont}_x(\breve{g}) = 1$ but $\mathrm{cont}_x(\breve{g}(\alpha)) \neq 1$.*

Example 9. *Suppose $\breve{g} = (12s + t)x + (s + 12t)z$. We have $\mathrm{cont}_x(\breve{g}) = \gcd(12s+t, s+12t) = 1$. But for $p = 11$ we obtain $cont_x(\breve{g} \bmod p) = s + t$. Hence $p = 11$ introduces an unlucky content and, for any prime p, the evaluation points $t = 0$ and $s = 0$ introduce unlucky contents.*

Suppose during sparse interpolation we choose our assumed form, g_f, based on an image which is computed modulo a prime (or evaluation point) which introduced an unlucky content, e.g. $p_1 = 11$ in our example. Then the assumed form g_f will have different terms in $x, z_1, ..., z_r, t_1, ..., t_k$ than in $\breve{g}$. This will, with high probability, result in an inconsistent linear system during sparse interpolation.

Suppose instead that our assumed form g_f is correct but it is a subsequent prime or evaluation point that introduces an unlucky content. This can lead to an under determined linear system.

EXAMPLE 10. *Consider $f_1 = f_2 = zx + t + 1$ where $m(z) = z^2 - t - 14$. Here $g = x + (t + 1)/(t + 14)z$ and hence $\breve{g} = (t+14)x + (t+1)z$ thus 13 introduces an unlucky content t. Suppose our first prime is $p_1 \neq 13$ and we obtain the correct assumed form $g_f = (At + B)x + (Ct + D)z$. Suppose our second prime is $p_2 = 13$ and we perform a sparse interpolation in t using $t = 1, 2, 3, \ldots$. Since g_f is not monic in x we will equate $g_f(t) = m_t\breve{g}(t)$ and solve for $A, B, C, D, m_1, m_2, \ldots$ with $m_1 = 1$. For $t = 1, 2, 3$ we obtain the following equations modulo 13.*

$$(A + B)x + (C + D)z \equiv m_1(x + z),$$
$$(2A + B)x + (2C + D)z \equiv m_2(x + z),$$
$$(3A + B)x + (3C + D)z \equiv m_3(x + z).$$

Equating coefficients of $x^i z^j$ we obtain the following linear system: $A+B = m_1$, $2A+B = m_2$, $3A+B = m_3$, $C+D = 1$, $2C + D = 1$, $3C + D = 1$. The reader may verify that this system, with $m_1 = 1$, is not determined, and also, adding further equations, e.g. from $t = 4$, does not make the system determined.

If our primes are sufficiently large and our evaluation points are chosen at random, then primes and evaluation points which introduce unlucky contents are rare. Because also, in $L[x]$, we cannot easily identify them in advance (it is not necessarily true that $\mathrm{lc}_x\breve{g}$ divides $\mathrm{lc}_x\breve{f_1}$) we will detect them through their effect. We will assume that if the linear system in sparse interpolation is not determined, an unlucky content is present, and we will design our algorithm to fail back to the point where the unlucky content was introduced.

It is also possible that the linear system is under determined because of the evaluation points chosen in sparse interpolation. For example, for $\breve{g} = x+(t^3-t)z$ with assumed form $g_f = x + (At^3 + B)z$, evaluation points $t = 0, 1, -1$ do not constrain the system. Thus when we mistakenly assume that this is because of an unlucky content, we will waste some useful work.

3. ALGORITHM SPARSEMODGCD

We now present the SparseModGcd algorithm. This modular GCD algorithm first calls subroutine M which computes the gcd in $L[x]$ from a number of images in $L_p[x]$. Subroutine P which is called by subroutine M computes the gcd in $L_p[x]$ using both dense and sparse interpolations. Finally subroutine S, which stands for Sparse Interpolation and is called by subroutine P, does the sparse interpolation.

Algorithm SparseModGcd
Input: f_1,$f_2 \in L[x]$ and $m_1, \ldots, m_r \in F[z_1, \ldots, z_r]$ where $F = \mathbb{Q}(t_1, \ldots, t_k)$ s.t. $\mathrm{cont}_x(g) = 1$.
Output: $\breve{g}$, where g is the monic gcd of f_1 and f_2 in $L[x]$.

```
1. Call Subroutine M with input $\breve{f}_1$, $\breve{f}_2$ and $\breve{m}_1$,...,$\breve{m}_r$.
```

Subroutine M
Input: f_1,$f_2 \in D[z_1, ..., z_r]/\langle m_1, ..., m_r\rangle [x]$ and $m_1, ..., m_r \in D[z_1, \ldots, z_r]$ were $D = \mathbb{Z}[t_1, \ldots, t_k]$.
Output: $\breve{g}$, where g is the monic gcd of f_1 and f_2.

```
1. Set $n = 1, G = 0$, and the assumed form $g_f = 0$.
2. Take a new prime $p_n$ that is not bad.
3. Let $g_n \in D_{p_n}[z_1, \ldots, z_r, x]$ be the output of subroutine P applied to $f_1$, $f_2$, $g_f$, $m_1$ mod $p$,..., $m_r$ mod $p$.
4. If $g_n$ = "ZeroDivisor" or $g_n$ = "Unlucky" or $g_n$ = "UnluckyContent" then go back to step 2.
5. If $g_n$ = "BadForm" then go back to step 1.
6. If $g_n = 1$ then return 1.
7. If $G = 0$ then set $G = g_n$, $M = p$ and go to step 11.
8. If $\deg_x(g_n) < \deg_x(G)$ then set $G = g_n$, $M = p$, and go to step 11. ( All previous primes were unlucky. )
9. If $\deg_x(g_n) > \deg_x(G)$ then go back to step 2.
   ( $p_n$ is an unlucky prime. )
10. Set $M = M \times p$ and combine $g_n$ with $\{g_1, \ldots, g_{n-1}\}$ using Chinese remaindering to obtain $G$ mod $M$.
11. Set $g_f = g_n$, $n = n + 1$.
12. Apply integer rational reconstruction to obtain $h$ satisfying $h \equiv G$ mod $M$. If this fails then go back to step 2.
13. Clear fractions in $\mathbb{Q}$: Set $h = \breve{h}$.
14. Trial division: if $h|f_1$ and $h|f_2$ then return $h$, otherwise, go back to step 2.
```

Subroutine P
Input: f_1, f_2, $g_f \in D_p[z_1, \ldots, z_r]/\langle m_1, \ldots, m_r\rangle [x]$ and $m_1, \ldots, m_r \in D_p[z_1, \ldots, z_r]$.
Output: Either $\breve{g}$ or "ZeroDivisor" or "Unlucky" or "BadForm" or "UnluckyContent" if the algorithm fails to compute the primitive associate of the monic gcd of f_1 and f_2 because of the bad choice of the prime or evaluation point.

```
1. If $k$ (the number of parameters) = 0 then output the result of the monic Euclidean algorithm applied to $f_1$,$f_2$. If a zero divisor is encountered then output "ZeroDivisor".
2. If the assumed form $g_f = 0$ then go to step 4.
3. Sparse Interpolation: ( we already know the form $g_f$ of the gcd from subroutine M. )
   Return $g_n \in D_p[z_1, \ldots, z_r, x]$, the output of subroutine S applied to $f_1$,$f_2$,$m_1, \ldots, m_r$ and $g_f$.
4. Choose $\alpha_1$ at random from $\mathbb{Z}_p$ that is not bad.
5. Let $g_1 \in \mathbb{Z}_p[t_1, \ldots, t_{k-1}][z_1, \ldots, z_r, x]$ be the output of subroutine P applied to $f_1$,$f_2$,$m_1, \ldots, m_r$ at $t_k = \alpha_1$ and assumed form $g_f = 0$.
6. If $g_1 = 1$ then return 1.
7. If $g_1 \in$ { "Unlucky", "UnluckyContent", "ZeroDivisor" } then return $g_1$.
8. Set $g_f = g_1$, $G = g_1$, $M = (t_k - \alpha_1)$, $n = 2$, $c = 1, d = 1, u = 1$.
9. Main Loop: Take a new evaluation point $\alpha_n$ at random from $\mathbb{Z}_p$ that is not bad.
10. Let $g_n \in \mathbb{Z}_p[t_1, \ldots, t_{k-1}][z_1, \ldots, z_r, x]$ be the output of subroutine S applied to $f_1$,$f_2$,$m_1, \ldots, m_r$ evaluated at $t_k = \alpha_n$ and $g_f$.
11. If $g_n$ = "BadForm" then return "BadForm".
12. If $g_n$ = "UnluckyContent" then set $c = c + 1$ and if $c > n$ return "UnluckyContent" else go to main loop.
13. If $g_n$ = "Unlucky" then set $u = u + 1$ and if $u > n$ then return "Unlucky", else go back to main loop.
14. If $g_n$ = "ZeroDivisor" then set $d = d + 1$ and if $d > n$ then return "ZeroDivisor", else go back to main loop.
```

```
15. If $g_n = 1$ then return 1.
16. If $\deg_x(g_n) < \deg_x(G)$ then set $G = g_n$, $M = t_k - \alpha_n$,
    $g_f = g_n$ then go to step 19.
    ( All previous evaluation points were unlucky )
17. If $\deg_x(g_n) > \deg_x(G)$ then go back to main loop.
    ( $\alpha_n$ is an unlucky evaluation point )
18. Set $M = M \times (t_k - \alpha_n)$ and Chinese remainder $g_n$
    with $\{g_1, \ldots, g_{n-1}\}$ to obtain $G \bmod M(t_k)$.
19. Set $n = n + 1$.
20. Apply rational function reconstruction to coefficients
    of $G$ to obtain $h \in \mathbb{Z}_p(t_k)\ [t_1, \ldots, t_{k-1}][z_1, \ldots, z_r, x]$
    s.t. $h \equiv G \mod M(t_k)$. If this fails, go back to main
    loop.
21. Clear fractions in $\mathbb{Z}_p(t_k)$: Set $h = \check{h}$.
22. Trial division: if $h|f_1$ and $h|f_2$ then return $h$, other-
    wise, go back to main loop.
```

Subroutine S
Input: $f_1,\ f_2,\ g_f \in D_p[z_1, \ldots, z_r]/\ \langle m_1, \ldots, m_r\rangle\ [x]$ and $m_1, \ldots, m_r \in D_p[z_1, \ldots, z_r]$ where $D_p = \mathbb{Z}_p[t_1, \ldots, t_k]$.
Output: Either $\check{g}$, the primitive associate of the monic gcd of f_1 and f_2, or "BadForm" or "ZeroDivisor" or "Unlucky" or "UnluckyContent."

```
1. If $k$ (the number of parameters) = 0 then call the monic
   Euclidean algorithm on $f_1$,$f_2$ and output the result.
2. Suppose the assumed form $g_f = \sum_i C_i T_i$ where $T_i$
   is a monomial in $(x, z_1, \ldots, z_r)$ and $C_i = \sum_j c_{ij} S_j$
   where $S_j$ is a monomial in parameters $t_1, \ldots, t_k$ with
   unknown $c_{ij}$.
3. Set $U$ to be the minimum number of images needed –
   see below. ( the algorithm uses one more image than
   $U$ to detect a wrong assumed form $g_f$. )
4. Set $z = 1$, $u = 1, n = 1$.
5. While $n \leq U + 1$ do
   5.1. Take a new random evaluation point $\alpha_n = (t_1 =
        a_1, \ldots, t_k = a_k)$ in $\mathbb{Z}_p^k$ which is not bad.
   5.2. Let $g_n$ be the output of the monic Euclidean al-
        gorithm applied to $f_1(\alpha_n)$, $f_2(\alpha_n)$, $m_1(\alpha_n)$,...,
        $m_r(\alpha_n)$.
   5.3. If $g_n$ = "ZeroDivisor" then set $z = z + 1$ and if
        $z > n$ then return "ZeroDivisor" else go back to
        step 5.1.
   5.4. If $\deg_x(g_n) > \deg_x(g_f)$ then set $u = u + 1$ and if
        $u > n$ then return "Unlucky" else go back to step
        5.1.
   5.5. If $\deg_x(g_n) < \deg_x(g_f)$ return "BadForm".
   5.6. If $g_n$ has terms in $(x, z_1, \ldots, z_r)$ not present in the
        assumed form $g_f$ then return "BadForm".
   5.7. Set $n = n + 1$.
6. Construct the system of $U + 1$ linear equations by
   equating $g_f(\alpha_n) = m_n\, g_n$ with $m_n$ unknown. Solve
   the linear system with $m_1 = 1$ to determine the $c_{ij}$'s.
7. If the system is inconsistent, return "BadForm".
8. If the system is under determined, return "Unlucky-
   Content".
9. Set $g_p = \sum_i (\sum_j c_{ij} S_j) T_i$.
10. Make $\mathrm{lc}_{x,t_1,\ldots,t_k}(g_p) = 1$ and return $g_p$.
```

Let n_i be the number of terms in the i'th coefficient C_i of the assumed form g_f. The minimum number of images needed in the sparse interpolation (with multiple scaling factors) to determine the $c'_{ij}s$ is $\max(M, \left\lceil \frac{N-1}{T-1} \right\rceil)$ where T is the number of monomials in g_f in $(x, z_1, \ldots, z_r)$, N is the total number of monomials in g_f and M is the maximum of the n_i's.

If the assumed form g_f in subroutine S is wrong, the linear system will most probably be inconsistent. It can, however, be consistent. If consistent, control passes back to subroutine P (or subroutine M) which will attempt rational function reconstruction (respectively, rational number reconstruction in subroutine M). If this succeeds the trial divisions prevent the algorithm from returning a wrong answer. Then subroutine P will call subroutine S at a new evaluation point with the same wrong assumed form. We argue that, provided p is sufficiently large, subroutine S will eventually get an inconsistent system and determine that the assumed form is wrong.

To treat zero divisors we use the same strategy used by Monagan and van Hoeij in ModGcd algorithm (see [3]). The variable d, in subroutine P, counts the number of times the Euclidean algorithm encounters a zero divisor. The case $d > n$ happens when the algorithm encounters a lot of zero divisors. This could relate to our choice of prime in subroutine M or a previous evaluation point in subroutine P. In this case subroutine P will quit. Note that if most evaluation points are good, and if subroutine P has already computed many good images, then the test $d > n$ prevents, with high probability, that an unlucky choice in Step 9 could cause a lot of useful work to be lost.

A similar strategy is also used in subroutine S for both zero divisors and unlucky primes to prevent useful work from being lost. The same strategy is also used in subroutine P to prevent useful work from being lost should the current evaluation point $t = \alpha_n$ in Step 10 introduce an unlucky content.

4. IMPLEMENTATION

Here we explain *trial division* which is a bottleneck in the implementation of SparseModGcd algorithm.

Trial Division

In Step 14 of subroutine M and Step 22 of subroutine P, the algorithm uses *trial division* to test whether it has computed the correct gcd. The only difference is that in subroutine P, the trial divisions take place in characteristic p. In [4] Monagan, van Hoeij presented an algorithm for doing trial divisions (in characteristic p) of polynomials in $\mathbb{Z}[z][x]$ modulo $m(z) \in \mathbb{Z}[z]$ which uses pseudo-division to avoid fractions and some gcds in $\mathbb{Z}$ to minimize growth of the integer coefficients. We can use the same idea for our algorithm, except that the coefficient ring is $D_p = \mathbb{Z}_p[t_1, \ldots, t_k]$ instead of $\mathbb{Z}$. The same algorithm can also be used for subroutine M with D_p replaced by D. Here we show how to extend it to treat multiple field extensions.

Algorithm Trial Division with Multiple Field Extensions
Input: $A, B \in D_p[z_1, ..., z_r]/\ \langle m_1, ..., m_r\rangle\ [x]$ and $m_1, ..., m_r \in D_p[z_1, ..., z_r]$, $B \neq 0$.
Output: True if $B|A$, False otherwise.

```
1. Set $m = \deg_x(A), n = \deg_x(B)$.
2. Set $d_1 = \deg_{z_1}(m_1), \ldots, d_r = \deg_{z_r}(m_r)$.
3. Set $l_b = \mathrm{lc}_x(B)$.
4. Set $l_{m_1} = \mathrm{lc}_{z_1}(m_1), \ldots, l_{m_r} = \mathrm{lc}_{z_r}(m_r)$.
5. Set $R = A$.
6. While $R \neq 0$ and $m \geq n$ do
  6.1. Set $l_R = \mathrm{lc}_x(R)$.
  6.2. Set $g = \gcd(\mathrm{cont}_{z_1,\ldots,z_r}(l_R), l_b)$ mod $p$.
  6.3. Set $l_R = l_R/g, s = l_b/g$.
  6.4. Set $t = l_R x^{m-n}$.
  6.5. Set $R = sR - tb$.
  6.6. For $i$ from 1 to $r$ do
         i. While $R \neq 0$ and $\deg_{z_i}(R) \geq d_i$ do
              A. Set $l_R = \mathrm{lc}_{z_i}(R)$.
              B. Set $g = \gcd(\mathrm{cont}_x(l_R), l_{m_i})$ mod $p$.
              C. Set $l_R = l_R/g$.
              D. Set $t = l_R z_i^{\deg_{z_i}(R)-d_i}$.
              E. Set $R = (l_{m_i}/g)R - tm_i$.
  6.7. Set $m = \deg_x(R)$.
7. If $R \neq 0$ then return False, otherwise, return True.
```

Note that $\deg_{z_j}(m_i) = 0$ if $j > i$. The outer loop reduces the degree of the remainder R in x. In the inner loops, for each i, the algorithm reduces the degree of R in z_i to be less than the degree of m_i in z_i.

4.1 Timings

We have compared Maple implementations of SparseModGcd, ModGcd and a primitive PRS algorithm (see appendix) on three problem sets. The input polynomials have a sparse gcd in the first and third set and a dense gcd in the second problem set. All timings are in CPU seconds and were obtained using Maple 10 on a 64 bit AMD Opteron CPU running Linux using 31.5 bit primes.

SPARSE-1

Let $m(z) = z^3 - (s+r)z^2 - (t+v)^2 z - 5 - 3u$. For $n = 1, 2, \ldots, 10$, let $f_1 = a \times g$ and $f_2 = b \times g$ where

$$g = sx_1^n + tx_2^n + ux_3^n + \sum_{j=1}^{4}\sum_{i=0}^{n-1} r_{i_j}^{(1)} z^{j-1} x_j^i + \sum_{w=[r,s,t,u,v]}\sum_{k=0}^{n} r_{w_k}^{(1)} w^k,$$

$$a = tx_1^n + ux_2^n + sx_3^n + \sum_{j=1}^{4}\sum_{i=0}^{n-1} r_{i_j}^{(2)} z^{j-1} x_j^i + \sum_{w=[r,s,t,u,v]}\sum_{k=0}^{n} r_{w_k}^{(2)} w^k,$$

$$b = ux_1^n + sx_2^n + tx_3^n + \sum_{j=1}^{4}\sum_{i=0}^{n-1} r_{i_j}^{(3)} z^{j-1} x_j^i + \sum_{w=[r,s,t,u,v]}\sum_{k=0}^{n} r_{w_k}^{(3)} w^k$$

and each $r_{j_k}^{(i)}$ is a positive random integer less than 100. Thus we have 10 gcd problems, all with one field extension $m(z)$, five parameters r, s, t, u and v and four variables x_1, x_2, x_3 and x_4. Each input polynomial is of degree $2n$ in the first three variables and $2n - 2$ in x_4. We wanted a data set of polynomials which are sparse but not too sparse.

Table 1 gives the running times for the three algorithms. In the first column, the numbers shown in parentheses are the percentages of the time which is spent computing univariate images of the gcd. Since the gcd g in this case is very sparse (g has $9n + 3$ terms and $\deg(g) = n$ in any of x_1, x_2 and x_3), a better performance is expected from SparseModGcd. The data demonstrates this clearly.

n	SparseModGcd	ModGcd	PPRS
1	0.38 (51.72%)	8.70	3.83
2	0.98 (62.15%)	114.78	> 2000
3	2.03 (67.06%)	879.26	NA
4	3.82 (73.40%)	> 2000	NA
5	6.58 (75.17%)	NA	NA
6	11.03 (77.76%)	NA	NA
7	17.31 (79.78%)	NA	NA
8	26.55 (81.31%)	NA	NA
9	39.02 (82.31%)	NA	NA
10	54.54 (82.34%)	NA	NA

Table 1: Timings (in CPU seconds) for SPARSE-1 (NA means not attempted)

DENSE-1

Let $m(z) = z^2 - sz - 3$. Suppose g, a and b are three randomly chosen polynomials in x_1, x_2, s and z of total degree n which are dense. That is the term $x_1^{d_1} x_2^{d_2} s^{d_3} z^{d_4}$ with $d_1 + d_2 + d_3 + d_4 \leq n$ is present in each of these three polynomials. So each of them has exactly $\sum_{i=0}^{n} \binom{i+4}{4}$ terms. For $n = 1, 2, \ldots, 10, 15$, let $f_1 = g \times a$ and $f_2 = g \times b$. Since the gcd g is dense, ModGcd algorithm is expected to perform better.

n	SMGCD	(EA,TDIV)	ModGcd	PPRS
1	0.031	(67.74%,6.45%)	0.029	0.002
2	0.070	(58.57%,21.43%)	0.058	0.384
3	0.150	(60.66%,24.67%)	0.141	367.234
4	0.308	(44.80%,34.42%)	0.307	> 2000
5	0.497	(40.44%,42.86%)	0.557	NA
6	0.901	(44.73%,43.40%)	1.272	NA
7	1.516	(38.19%,49.93%)	2.091	NA
8	2.443	(34.67%,53.70%)	3.244	NA
9	3.847	(30.93%,58.33%)	5.024	NA
10	6.120	(27.78%,63.71%)	7.437	NA
15	59.878	(29.48%,66.40%)	50.228	NA

Table 2: Timings (in CPU seconds) for DENSE-1 (NA means not attempted)

Table 2 shows the running times of the three algorithms for this set of problems. In the second column, the first number shown in parentheses is the percentage of the time spent computing univariate images of the gcd and the second is the percentage spent doing trial division. The reader may see that SparseModGcd is very competitive with ModGcd even though the gcds are completely dense.

SPARSE-2

Let $T = [s, t, u, s, t, u, \ldots]$ and $Z = [z_1, z_2, z_1, z_2, \ldots]$. Suppose we have

$$m_1(z_1) = z_1^2 - (t^2 + s)z_1 - 5 - 3u, \; m_2(z_2) = z_2^2 + (s - z_1)z_2 + 3t - u,$$

$$g = sx^n + \sum_{i=0}^{n-1} iT_{i+1} Z_{i+1} x^i,$$

$$a = tx^n + \sum_{i=0}^{n-1} iT_{i+2} Z_{i+2} x^i,$$

$$b = ux^n + \sum_{i=0}^{n-1} iT_{i+3}Z_{i+3}x^i.$$

Again for $n = 1, 2, \ldots, 10$, let $f_1 = a \times g$ and $f_2 = b \times g$. So we have 10 gcd problems, with two field extensions m_1 and m_2, three parameters s, t and u and one variable x. Each input polynomial is of degree $2n$ in x.

n	SparseModGcd	PPRS
1	0.096 (87.50%)	0.004
2	0.229 (84.28%)	194.540
3	0.361 (87.53%)	> 2000
4	0.510 (88.82%)	NA
5	0.695 (90.37%)	NA
6	0.921 (91.42%)	NA
7	1.180 (90.34%)	NA
8	1.493 (90.69%)	NA
9	1.811 (93.04%)	NA
10	2.172 (93.42%)	NA

Table 3: Timings (in CPU seconds) for SPARSE-2 (NA means not attempted)

The timings for SparseModGcd and primitive PRS algorithms for this problem set are shown in Table 3. In the first column, the numbers shown in parentheses are the percentages of the time spent computing univariate images of the gcd. Since ModGcd does not support multiple field extensions, there are no timings for ModGcd. The data here demonstrates the superiority of SparseModGcd algorithm.

5. CONCLUDING REMARKS

Suppose $L = F[z]/\langle m\rangle$ and m is reducible over F. Thus L is not a field. It is a commutative ring with zero divisors. Then it may still be desirable to compute a gcd (if it exists) over L. If the Euclidean algorithm when applied to f_1 and f_2 would encounter a zero divisor (a factor of $m(z)$) then our algorithm, will most likely go into an infinite loop – it will most likely repeatedly encounter an image of the zero divisor.

In principle, one may modify our algorithm to interpolate (using sparse interpolation) the zero divisor. Since one will not know whether the zero divisor exists over $\mathbb{Q}$ or is caused by an unlucky choice of prime or evaluation point, one simultaneously interpolates the gcd and/or zero divisor. Subroutines M and P would terminate when either the interpolated zero divisor divides $m(z)$ or the interpolated gcd divides f_1 and f_2.

The algorithm of Dahan *et al.* is also a sparse GCD algorithm. It uses Hensel lifting instead of sparse interpolation. We did not consider using Hensel lifting here for GCD computation over L because of the following problem. To use Hensel lifting one would need to compute, and possibly also factor, $\alpha(t) \in \mathbb{Z}[t_1, ..., t_k]$, a multiple of the leading coefficient of $\check{g}$. For an algebraic function field L computing $\alpha(t)$ requires us to invert the leading coefficients of $\check{f}_1$ and $\check{f}_2$ (or compute a resultant) which may introduce a severe expression swell.

Another advantage of sparse interpolation over Hensel lifting is that it is easy to parallelize the most time consuming part. In our implementation most of the time is either spent reducing (evaluating) $\check{f}_1$, $\check{f}_2$, and $\check{m}_1, \ldots, \check{m}_r$ modulo $I = \langle t_1 - \alpha_1, \ldots, t_r - \alpha_r\rangle$ modulo a prime and/or spent in the Euclidean algorithm when computing the gcd of the images $\check{f}_1$ and $\check{f}_2$ modulo $\check{m}_1, \ldots, \check{m}_r$ modulo I. If we need n images to interpolate the i'th parameter t_i, after computing the first image, the remaining $n - 1$ images can be computed (using sparse interpolation) in parallel. And if we need m images for each sparse interpolation, these too can be computed in parallel.

6. REFERENCES

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

[2] X. Dahan, M. Moreno Maza, E. Schost, W. Wu and Y. Xie, Lifting Techniques for Triangular Decompositions. *Proceedings of ISSAC '05*, ACM Press, pp. 108–115, 2005.

[3] Mark van Hoeij and Michael Monagan. Algorithms for Polynomial GCD Computation over Algebraic Function Fields, *Proceedings of ISSAC '04*, ACM Press, pp. 297–304, 2004.

[4] Mark van Hoeij and Michael Monagan, A modular GCD algorithm over number fields presented with multiple extensions. *Proceedings of ISSAC '02*, ACM Press, pp. 109–116, 2002.

[5] J. de Kleine, M. Monagan and A. Wittkopf, Algorithms for the non-monic case of the sparse modular GCD algorithm. *Proceedings of ISSAC '05*, ACM Press, pp. 124–131, 2005.

[6] Marc Moreno Maza and Renaud Rioboo. Polynomial Gcd Computations over Towers of Algebraic Extensions. *Proceedings of AAECC-11*, Springer-Verlag, pp. 365–382, 1995.

[7] Michael Monagan, Maximal Quotient Rational Reconstruction: An Almost Optimal Algorithm for Rational Reconstruction. *Proceedings of ISSAC '04*, ACM Press, 243–249, 2004.

[8] P. S. Wang, M. J. T. Guy, J. H. Davenport. Early detection of true factors in Univariate Polynomial Factorization. *Proceedings of EUROCAL '83*, Springer-Verlag LNCS **162**, pp. 225–235.

[9] Richard Zippel, Probabilistic algorithms for sparse polynomials. *Proceedings of EUROSAM '79*, Springer-Verlag, pp. 216–226, 1979.

Appendix

We describe the algorithm and give Maple code for the primitive fraction-free algorithm that we used to compute a gcd in $L[x]$ for comparison with SparseModGcd algorithm. We think of computing in L as computing modulo the triangular set $M = \{m_1, \ldots, m_r\}$. To avoid fractions, we first set $f_1 := \check{f}_1$, $f_2 := \check{f}_2$ and $M := \{\check{m}_1, \ldots, \check{m}_r\}$. Now suppose we apply the Euclidean algorithm to compute the gcd of f_1 and f_2 modulo M. We would divide f_1 by f_2. In the ordinary division algorithm we would invert the leading coefficient u of the divisor f_2, an algebraic function. The coefficients of the inverse of u would have fractions in $F = \mathbb{Q}(t_1, \ldots, t_k)$.

To avoid fractions here we compute instead v, a quasi-inverse of u, an element of $D[z_1, \ldots, z_r]$ satisfying $vu = c$ for some constant $c \in D = \mathbb{Z}[t_1, \ldots, t_k]$. Now we divide f_1 by vf_2 using pseudo division (mod M). And we make the pseudo remainder "primitive", i.e., we compute and cancel out any common factor in D from the coefficients.

To compute the quasi-inverse v we first apply the (extended) Euclidean algorithm to $\check{m}_r$ and u viewing them as elements of $K[z_r]$ where $K = F[z_1, ..., z_{r-1}]/\langle m_1, ..., m_{r-1}\rangle$. Again, we want to avoid fractions so we use pseudo-division. We perform pseudo-division in $D[z_1, \ldots, z_{r-1}][z_r]$. We obtain s, t, c satisfying

$$s m_r + t u = c \quad \text{where} \quad c \in D[z_1, \ldots, z_{r-1}].$$

Here c does not involve z_r but may involve $z_1, \ldots, z_{r-1}$. Next we recursively compute a quasi-inverse w of c satisfying $wc \in D$ and hence $v = wt$ is a quasi-inverse of u and we reduce wt modulo M using pseudo-division. Here is the algorithm in Maple code.

```
macro( `mod` = MOD );
MOD := proc(f,M,Z) local r,i;
  r := expand(f);
  for i to nops(M) do r := prem(r,M[i],Z[i]) od;
  r;
end:

# This uses the reduced PRS
QuasiInv := proc(x,M,Z)
local u,r0,r1,r2,t0,t1,t2,pq,mu,i,z,beta;
  if M=[] then return 1 fi;
  u := primpart(x,Z);
  r0,r1,t0,t1,beta := M[1],u,0,1,1;
  z := Z[1]; # main variable
  while degree(r1,z)>0 do
    r2 := prem(r0,r1,z,'mu','pq');
    divide(r2,beta,'r2');
    divide(mu*t0 - pq*t1,beta,'t2');
    r0,r1,t0,t1,beta := r1,r2,t1,t2,mu;
  od;
  if r1=0 then error "inverse does not exist" fi;
  if nops(M)>1 then
    r1 := r1 mod (M,Z);
    t1 := t1 mod (M,Z);
    i := QuasiInv(r1,M[2..-1],Z[2..-1]);
    t1 := i*t1 mod (M,Z):
  fi;
  primpart(t1,Z);
end:

PrimitivePRS := proc(f1,f2,x,M,Z)
local xZ,i,r0,r1,r2;
  xZ := [x,op(Z)];
  r0 := primpart(f1,xZ);
  r1 := primpart(f2,xZ);
  while r1 <> 0 do
    i := QuasiInv(lcoeff(r1,x),M,Z);
    r1 := primpart(i*r1 mod (M,Z), xZ);
    r2 := prem(r0,r1,x) mod (M,Z);
    r2 := primpart(r2,xZ);
    r0,r1 := r1,r2;
  od;
  r0;
end:
```