A Fast Parallel Sparse Polynomial GCD Algorithm*

JIAXIONG HU, Simon Fraser University, Canada
MICHAEL MONAGAN, Simon Fraser University, Canada

We present a parallel GCD algorithm for sparse multivariate polynomials with integer coefficients. The algorithm combines a Kronecker substitution with a Ben-Or/Tiwari sparse interpolation modulo a smooth prime to determine the support of the GCD. We have implemented our algorithm in Cilk C. We compare it with Maple and Magma's serial implementations of Zippel's GCD algorithm.

CCS Concepts: • **Computing methodologies → Algebraic algorithms**;

Additional Key Words and Phrases: Polynomial GCD computation, sparse polynomial interpolation

ACM Reference Format:
Jiaxiong Hu and Michael Monagan. 2017. A Fast Parallel Sparse Polynomial GCD Algorithm. *J. ACM* 1, 1 (December 2017), 41 pages. https://doi.org/0000001.0000001

1 INTRODUCTION

Let A and B be two polynomials in $\mathbb{Z}[x_0, x_1, \ldots, x_n]$. In this paper we present a modular GCD algorithm for computing $G = \gcd(A, B)$ the greatest common divisor of A and B which is designed for sparse A and B. We will compare our algorithm with Zippel's sparse modular GCD algorithm from [Zippel 1979]. Zippel's algorithm is the main GCD algorithm currently used by the Maple, Magma and Mathematica computer algebra systems for polynomials in $\mathbb{Z}[x_0, x_1, \ldots, x_n]$.

Multivariate polynomial GCD computation was a central problem in Computer Algebra in the 1970's and 1980's. Whereas classical algorithms for polynomial multiplication and exact division are sufficient for many inputs, this is not the case for polynomial GCD computation. Euclid's algorithm, and variant's of it such as the reduced PRS algorithm [Collins 1967] and the subresultant PRS algorithm [Brown and Traub 1971], result in an intermediate expression swell of size exponential in n when applied to sparse multivariate polynomials. This renders these algorithms useless even for inputs of a very modest size. GCD algorithms which avoid this intermediate expression swell include the dense

*This work was supported by NSERC of Canada and Maplesoft

Authors' addresses: Jiaxiong Hu, Department of Mathematics, Simon Fraser University, Burnaby, BC, V5A 1S6, Canada, jha107@sfu.ca; Michael Monagan, Department of Mathematics, Simon Fraser University, Burnaby, BC, V5A 1S6, Canada, mmonagan@sfu.ca.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. Copyrights for components of this work owned by others than ACM must be honored. Abstracting with credit is permitted. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee. Request permissions from permissions@acm.org.
© 2017 Association for Computing Machinery.
0004-5411/2017/12-ART $15.00
https://doi.org/0000001.0000001

modular GCD algorithm [Brown 1971], the the EEZ-GCD algorithm [Wang 1980], the sparse modular algorithm [Zippel 1979], the heuristic GCD algorithm [Char et al. 1989], and the black-box algorithm [Kaltofen and Trager 1990]. For the interested reader, Chapter 7 of [Geddes et al. 1992] provides a description of the algorithms in [Brown 1971; Char et al. 1989; Wang 1980; Zippel 1979].

Let $A = \sum_{i=0}^{d_A} a_i x_0^i$, $B = \sum_{i=0}^{d_B} b_i x_0^i$ and $G = \sum_{i=0}^{d_G} c_i x_0^i$ where $d_A > 0$, $d_B > 0$ and the coefficients a_i, b_i and c_i are in $\mathbb{Z}[x_1, \ldots, x_n]$. Our GCD algorithm first computes and removes contents, that is computes $\mathrm{cont}(A, x_0) = \gcd(a_i)$ and $\mathrm{cont}(B, x_0) = \gcd(b_i)$. These GCD computations in $\mathbb{Z}[x_1, x_2, \ldots, x_n]$ are computed recursively.

Let $\bar{A} = A/G$ and $\bar{B} = B/G$ be the cofactors of A and B respectively. Let $\#A$ denote the number of terms in A and let $\mathrm{Supp}(A)$ denote the set of monomials appearing in A. Let $\mathrm{LC}(A)$ denote the leading coefficient of A taken in x_0. Let $\Gamma = \gcd(LC(A), LC(B)) = \gcd(a_{d_A}, b_{d_B})$. Since $LC(G)|LC(A)$ and $\mathrm{LC}(G)|LC(B)$ it must be that $\mathrm{LC}(G)|\Gamma$ thus $\Gamma = LC(G)\Delta$ for some polynomial $\Delta \in \mathbb{Z}[x_1, \ldots, x_n]$.

Example 1.1. If $G = x_1 x_0^2 + x_2 x_0 + 3$, $\bar{A} = (x_2 - x_1)x_0 + x_2$ and $\bar{B} = (x_2 - x_1)x_0 + x_1 + 2$ we have $\#G = 3$, $\mathrm{Supp}(G) = \{x_1 x_0^2, x_2 x_0, 1\}$, $\mathrm{LC}(G) = x_1$, $\Gamma = x_1(x_2 - x_1)$, and $\Delta = x_2 - x_1$.

We provide an overview of our GCD algorithm. Let $H = \Delta \times G$ and $h_i = \Delta \times c_i$ so that $H = \sum_{i=0}^{d_G} h_i x_0^i$. Our algorithm will compute H not G. After computing H it must then compute $\mathrm{cont}(H, x_0) = \gcd(h_i) = \Delta$ and divide H by Δ to obtain G. We compute H modulo a sequence of primes $p_1, p_2, \ldots,$ and recover the integer coefficients of H using Chinese remaindering. The use of Chinese remaindering is standard. Details may be found in [Brown 1971; Geddes et al. 1992]. Let H_1 be the result of computing $H \mod p_1$. For the remaining primes we use the sparse interpolation approach of Zippel [Zippel 1979] which assumes $\mathrm{Supp}(H_1) = \mathrm{Supp}(H)$. From now on we focus on the computation of $H \mod p_1$.

To compute $H \bmod p$ the algorithm will pick a sequence of points $\beta_1, \beta_2, \ldots$ from $\mathbb{Z}_p^n$, compute monic images

$$g_j := \gcd(A(x_0, \beta_j), B(x_0, \beta_j)) \in \mathbb{Z}_p[x_0]$$

of G, in parallel, then multiply g_j by the scalar $\Gamma(\beta_j) \in \mathbb{Z}_p$. Because the scaled image $\Gamma(\beta_j) \times g_j(x_0)$ is an image of a polynomial, H, we can use polynomial interpolation to interpolate each coefficient $h_i(x_1, \ldots, x_n)$ of H from the coefficients of the scaled images.

Let $t = \max_{i=0}^{d_G} \#h_i$. The parameter t measures the sparsity of H. Let $d = \max_{i=1}^{n} \deg_{x_i} H$ and $D = \max_{i=0}^{d_G} \deg h_i$. The cost of sparse polynomial interpolation algorithms is determined mainly by the number of points $\beta_1, \beta_2, \ldots$ needed and the size of the prime p needed. These depend on n, t, d and D. Table 1 below presents data for several sparse polynomial interpolation algorithms. In Table 1 p_n denotes the n'th prime which has size $O(\log n \log\log n)$ bits. Other sparse interpolation algorithms, not directly applicable to the GCD problem, are mentioned in the concluding remarks.

To get a sense for how large the prime needs to be for the different algorithms in Table 1 we include data for the following **benchmark problem:** Let $G, \bar{A}, \bar{B}$ have nine variables ($n = 8$), have degree $d = 20$ in each variable, and have total degree $D = 60$ (to better reflect real problems). Let G have 10,000 terms with $t = 1000$. Let $\bar{A}$ and $\bar{B}$ have 100 terms so that $A = G\bar{A}$ and $B = G\bar{B}$ have about one million terms.

	#points	size of p	benchmark
Zippel [1979]	$O(ndt)$	$p > 2nd^2t^2$	$= 6.4 \times 10^9$
BenOr/Tiwari [1988]	$O(t)$	$p > p_n^D$	$= 5.3 \times 10^{77}$
Monagan/Javadi [2010]	$O(nt)$	$p > nDt^2$	$= 4.8 \times 10^8$
Murao/Fujise [1996]	$O(t)$	$p > (d+1)^n$	$= 3.7 \times 10^{10}$

Table 1. Some sparse interpolation algorithms

Notes: Zippel's sparse interpolation algorithm [Zippel 1979] is probabilistic. It was developed for polynomial GCD computation and implemented in Macsyma by Zippel. Rayes, Wang and Weber parallelized parts of it in [Rayes et al. 1994] for shared memory computers. Kaltofen and Lee showed in [Kaltofen et al. 2000] how to modify Zippel's algorithm so that it will work effectively for primes much smaller than $2nd^2t^2$.

The Ben-Or/Tiwari algorithm [Ben-Or and Tiwari 1988] is deterministic. The primary disadvantage of the Ben-Or/Tiwari algorithm is the size of the prime. [Javadi and Monagan 2010] modify the Ben-Or/Tiwari algorithm to work for a smaller prime but using $O(nt)$ points.

[Murao and Fujise 1996]'s method is a modification of the Ben-Or/Tiwari algorithm which computes discrete logarithms in the cyclic group $\mathbb{Z}_p^*$. We will refer to this method as the "discrete logs" method. We give details for it in Section 1.2. The advantage over the Ben-Or/Tiwari algorithm is that the prime size is $O(n \log d)$ bits instead of $O(D \log n \log \log n)$ bits.

In a GCD algorithm that uses interpolation from values, not all evaluation points can be used. Let $\beta_j \in \mathbb{Z}_p^n$ be an evaluation point. If $\gcd(\bar{A}(x_0, \beta_j), \bar{B}(x_0, \beta_j)) \neq 1$ then β_j is said to be *unlucky* and this image cannot be used to interpolate H. Section 1.4 characterizes which evaluation points are unlucky and describes how they can be detected. In Zippel's algorithm, where the β_j are chosen at random from $\mathbb{Z}_p^n$, unlucky β_j, once identified, can simply be skipped. This is not the case for the evaluation point sequences used by the Ben-Or/Tiwari algorithm and the discrete logs method. In Section 1.5, we modify these point sequences to handle unlucky evaluation points.

To reduce the probability of encountering unlucky evaluation points, the prime p may need to be larger than that shown in Table 1. Our modification for the discrete logarithm sequence increases the size of p which negates much of its advantage. This led us to consider using a Kronecker substitution K_r on $x_1, x_2, \ldots, x_n$ to map the GCD computation into a bivariate computation in $\mathbb{Z}_p[x_0, y]$. Some Kronecker substitutions result in all evaluation points being unlucky so they cannot be used. We call these Kronecker substitutions *unlucky*. In Section 2 we show (Theorem 2.5) that there are only finitely many of them and how to detect them so that a larger Kronecker substitution may be tried.

If a Kronecker substitution is not unlucky there can still be many unlucky evaluation points because the degree of the resulting polynomials $K_r(A)$ and $K_r(B)$ in y is exponential in n. In order to avoid unlucky evaluation points one may simply choose the prime $p \gg \max(\deg_y(K_r A), \deg_y(K_r B))$, which is what we do for our "simplified" version of our GCD algorithm. But this may mean p is not a machine prime which will significantly increase the cost of all modular arithmetic in $\mathbb{Z}_p$ as multi-precision arithmetic is needed. However, it is well known in the computer algebra research community that unlucky evaluation points are infact rare. This prompted us to investigate the distribution of the unlucky evaluation points. Our next contribution (Theorem 2.13) is a result for the expected number of unlucky evaluations. This theorem justifies our "faster" version of our GCD algorithm which first tries a smaller prime.

In Section 3 we assemble a "Simplified Algorithm" which is a Las Vegas GCD algorithm. It first applies a Kronecker substitution to map the GCD computation into $\mathbb{Z}[x_0, y]$. It then chooses p randomly from a large set of smooth primes and computes $H \mod p$ using sparse interpolation in y then uses further primes and Chinese remaindering to recover the integer coefficients in H. The algorithm chooses a Kronecker substitution large enough to be a priori not unlucky and assumes a term bound $\tau \geq \max \#h_i$ is given. These assumptions lead to a much simpler algorithm.

In Section 4, we relax the term bound requirement and we first try a Kronecker substitution just large enough to recover H. This complicates significantly the GCD algorithm. In Section 4 we present a heuristic GCD algorithm which we can prove always terminates and outputs $H \bmod p$. The heuristic algorithm will usually be much faster than the simplified algorithm but it can, in theory, fail several times before it finds a Kroenecker substitution K_r, a sufficiently large prime p, and evaluation points β_j which are all good.

We have implemented our algorithm in C and parallelized it using Cilk C. We did this initially for 31 bit primes then for 63 bit primes and then for 127 bit primes to handle polynomials in more variables. The first timing results revealed that almost all the time was spent in evaluating $A(x_0, \beta_j)$ and $B(x_0, \beta_j)$ and not interpolating H. In Section 5 we describe an improvement for evaluation and how we parallelized it.

In Section 6 we compare our new algorithm with the C implementations of Zippel's algorithm in Maple and Magma. The timing results are very promising. For our benchmark problem, Maple takes 22,111 seconds, Magma takes 1,611 seconds. and our new algorithm takes 4.67 seconds on 16 cores.

If $\#\Delta > 1$ then the number of terms in H may be (much) larger than the number of terms in G. Sections 5.2 and 5.3 describe two practical improvements to reduce $\#\Delta$ and hence reduce t. The second improvement reduces the time for our benchmark problem from 4.67 seconds to 0.652 seconds on 16 cores.

1.1 Some notation and results

The proofs in the paper make use of properties of the Sylvester resultant, the Schwartz-Zippel Lemma and require bounds for the size of the integer coefficients appearing in certain polynomials. We state these results here for later use.

Let $f = \sum_{i=1}^{t} a_i M_i$ where $a_i \in \mathbb{Z}$, $a_i \neq 0$, $t \geq 0$ and M_i is a monomial in n variables $x_1, x_2, \ldots, x_n$. We denote by $\deg f$ the total degree of f, $\deg_{x_i} f$ the degree of f in x_i, and $\#f$ the number of terms of f. We need to bound the size of the the integer coefficients of certain polynomials. For this purpose let $\|f\|_1 = \sum_{i=1}^{t} |a_i|$ be the one-norm of f and $\|f\| = \max_{i=1}^{t} |a_i|$ be the height of f. For a prime p, let ϕ_p denote the modular mapping $\phi_p(f) = f \mod p$.

Lemma 1.2. *[Schwartz 1980; Zippel 1979] Let F be a field and $f \in F[x_1, x_2, \ldots, x_n]$ be non-zero with total degree D and let $S \subset F$. If β is chosen at random from S^n then* $\mathrm{Prob}[f(\beta) = 0] \leq \frac{D}{|S|}$. *Hence if $R = \{\beta | f(\beta) = 0\}$ then $|R| \leq D|S|^{n-1}$.*

Lemma 1.3. *[Gelfond 1952] Lemma II page 135. Let f be a polynomial in $\mathbb{Z}[x_1, x_2, \ldots, x_n]$ and let d_i be the degree of f in x_i. If g is any factor of f over $\mathbb{Z}$ then $\|g\| \leq \mathrm{e}^{d_1+d_2+\cdots+d_n} \|f\|$ where $\mathrm{e} = 2.71828$.*

Let A be an $m \times m$ matrix with entries $A_{i,j} \in \mathbb{Z}$. Hadamard's bound $H(A)$ for $|\det(A)|$ is

$$|\det A| \leq \prod_{i=1}^{m} \sqrt{\sum_{j=1}^{m} A_{i,j}^2} = H(A).$$

Lemma 1.4. *[Goldstein and Graham 1974] Let A be an $m \times m$ matrix with entries $A_{i,j} \in \mathbb{Z}[y]$. Let B be the $m \times m$ integer matrix with $B_{i,j} = \|A_{i,j}\|_1$. Then $\|\det A\| \leq H(B)$.*

For polynomials $A = \sum_{i=0}^{s} a_i x_0^i$ and $B = \sum_{i=0}^{t} b_i x_0^i$, Sylvester's matrix is the following $s + t$ by $s + t$ matrix

$$S = \begin{bmatrix} a_s & 0 & & 0 & b_t & 0 & & 0 \\ a_{s-1} & a_s & & 0 & b_{t-1} & b_t & & 0 \\ \vdots & a_{s-1} & \ddots & 0 & \vdots & b_{t-1} & \ddots & 0 \\ a_1 & \vdots & & a_s & b_1 & \vdots & & b_t \\ a_0 & a_1 & & a_{s-1} & b_0 & b_1 & & b_{t-1} \\ 0 & a_0 & & \vdots & 0 & b_0 & & \vdots \\ 0 & 0 & \ddots & a_1 & 0 & 0 & \ddots & b_1 \\ 0 & 0 & & a_0 & 0 & 0 & & b_0 \end{bmatrix}. \tag{1}$$

where the coefficients of A are repeated in the first t columns and the coefficients of B are repeated in the last s columns. The Sylvester resultant of the polynomials A and B in x, denoted $\mathrm{res}_x(A, B)$, is the determinant of Sylvester's matrix. We gather the following facts about it into Lemma 1.5 below.

Lemma 1.5. *Let D be any integral domain and let A and B be two polynomials in $D[x_0, x_1, \ldots, x_n]$ with $s = \deg_{x_0} A > 0$ and $t = \deg_{x_0} B > 0$. Let $a_s = LC(A)$, $b_t = LC(B)$, $R = \mathrm{res}_{x_0}(A, B)$, $\alpha \in D^n$ and p be a prime. Then*

(i) R is a polynomial in $D[x_1, \ldots, x_n]$,
(ii) $\deg R \leq \deg A \deg B$ (Bezout bound) and
(iii) $\deg_{x_i} R \leq t \deg_{x_i} A + s \deg_{x_i} B$ for $1 \leq i \leq n$.

If D is a field and $a_s(\alpha) \neq 0$ and $b_t(\alpha) \neq 0$ then
(iv) $\mathrm{res}_{x_0}(A(x_0, \alpha), B(x_0, \alpha)) = R(\alpha)$ and
(v) $\deg_{x_0} \gcd(A(x_0, \alpha), B(x_0, \alpha)) > 0 \iff \mathrm{res}_{x_0}(A(x_0, \alpha), B(x_0, \alpha)) = 0$.

If $D = \mathbb{Z}$ and $\phi_p(a_s) \neq 0$ and $\phi_p(b_t) \neq 0$ then
(vi) $\mathrm{res}_{x_0}(\phi_p(A), \phi_p(B)) = \phi_p(R)$ and
(vii) $\deg_{x_0} \gcd(\phi_p(A), \phi_p(B)) > 0 \iff \mathrm{res}_{x_0}(\phi_p(A), \phi_p(B)) = 0$.

Proofs of (i), (ii), (iv) and (v) may be found in Ch. 3 and Ch. 6 of [Cox et al. 1991]. In particular the proof in Ch. 6 of [Cox et al. 1991] for (ii) for bivariate polynomials generalizes to the multivariate case. Note that the condition on α that the leading coefficients a_s and b_t do not vanish means that the dimension of Sylvester's matrix for $A(x_0, \alpha)$ and $B(x_0, \alpha)$ is the same as that for A and B which proves (v). The same argument used to prove (iv) and (v) works for (vi) and (vii). To prove (iii) we have

$$\deg_{x_i} \det S \leq \sum_{c \in columns(S)} \max_{f \in c} \deg_{x_i} f = \sum_{j=1}^{t} \deg_{x_i} A + \sum_{j=1}^{s} \deg_{x_i} B.$$

1.2 Ben-Or Tiwari Sparse Interpolation

Let $C(x_1, \ldots, x_n) = \sum_{i=1}^{t} a_i M_i$ where $a_i \in \mathbb{Z}$ and M_i are monomials in $(x_1, \ldots, x_n)$. In our context, C represents one of the coefficients of $H = \Delta G$ we wish to interpolate. Let $D = \deg C$ and let $d = \max_{i=1}^{n} \deg_{x_i} C$ and let p_n denote the n'th prime. Let

$$v_j = C(2^j, 3^j, 5^j, \ldots, p_n^j) \text{ for } j = 0, 1, \ldots, 2t-1.$$

The Ben-Or/Tiwari sparse interpolation algorithm [Ben-Or and Tiwari 1988] interpolates $C(x_1, x_2, \ldots, x_n)$ from the $2t$ points v_j. Let $m_i = M_i(2, 3, 5, \ldots, p_n) \in \mathbb{Z}$ and let $\lambda(z) = \prod_{i=1}^{t}(z - m_i) \in \mathbb{Z}[z]$. The algorithm proceeds in 5 steps.

1 Compute $v_j = C(2^j, 3^j, 5^j, \ldots, p_n^j)$ for $j = 0, 1, \ldots, 2t-1$.
2 Compute $\lambda(z)$ from v_j using the Berlekamp-Massey algorithm [Massey 1969] or the Euclidean algorithm [Atti et al. 2006; Sugiyama et al. 1975].
3 Compute the integer roots m_i of $\lambda(z)$.
4 Factor the integers m_i using trial division by $2, 3, \ldots, p_n$ from which we obtain M_i. For example, for $n = 3$, if $m_i = 45000 = 2^3 3^2 5^4$ then $M_i = {x_1}^3 {x_2}^2 {x_3}^4$.
5 Solve the following $t \times t$ linear system for the unknown coefficients a_i in $C(x_1, \ldots, x_n)$.

$$V\,a = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ m_1 & m_2 & \ldots & m_t \\ {m_1}^2 & {m_2}^2 & \ldots & {m_t}^2 \\ \vdots & \vdots & \vdots & \vdots \\ {m_1}^{t-1} & {m_2}^{t-1} & \ldots & m_t^{t-1} \end{bmatrix} \begin{bmatrix} a_1 \\ a_2 \\ a_3 \\ \vdots \\ a_t \end{bmatrix} = \begin{bmatrix} v_0 \\ v_1 \\ v_2 \\ \vdots \\ v_{t-1} \end{bmatrix} = b \tag{2}$$

The matrix V above is a transposed Vandermonde matrix. Recall that

$$\det V = \det V^T = \prod_{1 \leq j < k \leq t} (m_j - m_k).$$

Since the monomial evaluations $m_i = M_i(2, 3, 4, \ldots, p_n)$ are distinct it follows that $Va = b$ has a unique solution. The linear system $Va = b$ can be solved in $O(t^2)$ arithmetic operations (see [Zippel 1990]). Note, the master polynomial $P(Z)$ in [Zippel 1990] is $\lambda(z)$.

Notice that the largest integer in $\lambda(z)$ is the constant term $\Pi_{i=1}^{t} m_i$ which is at most p_n^{Dt} hence of size $O(tD \log n \log\log n)$ bits. Moreover, in [Kaltofen et al. 1990], Kaltofen, Lakshman and Wiley noticed that a severe expression swell occurs if either the Berlekamp-Massey algorithm or the Euclidean algorithm is used to compute $\lambda(z)$ over $\mathbb{Q}$. For our purposes, because we want to interpolate H modulo a prime p, we run Steps 2, 3, and 5 modulo p. Provided we pick $p > \max_{i=1}^{t} m_i \leq p_n^D$ the integers m_i remain unique modulo p and we recover the monomials $M_i(x_1, \ldots, x_n)$ in Step 4 and the linear system in Step 5 has a unique solution modulo p. For Step 3, the roots of $\lambda(z) \in \mathbb{Z}_p[z]$ can be found using Berlekamp's algorithm [Berlekamp 1970] which has classical complexity $O(t^2 \log p)$.

In [Ben-Or and Tiwari 1988], Ben-Or and Tiwari assume a sparse term bound $T \geq t$ is known, that is, we are given some T such that $t \leq T \ll (d+1)^n$ and in Step 1 we may compute $2T$ evaluations in parallel. In practice such a bound on t may not known in advance so the algorithm needs to be modified to also determine t. For p sufficiently large, if we compute $\lambda(z)$ after $j = 2, 4, 6, \ldots$ points, we will see $\deg \lambda(z) = 1, 2, 3, \ldots, t-1, t, t, t, \ldots$ with high probability. Thus we may simply wait until the degree of $\lambda(z)$ does not change. This problem is first discussed by Kaltofen, Lee and Lobo in [Kaltofen et al. 2000]. We will return to this in Section 4.1.

Steps 2, 3, and 5 may be accelerated with fast multiplication. Let $M(t)$ denote the cost of multiplying two polynomials of degree t in $\mathbb{Z}_p[t]$. The fast Euclidean algorithm can be used to accelerate Step 2. It has complexity $O(M(t) \log t)$. See Ch. 11 of [von zur Gathen and Gerhard 1999]. Computing the roots of $\lambda(z)$ in Step 3 can be done in $O(M(t) \log t \log(pt))$. See Corollary 14.16 of [von zur Gathen and Gerhard 1999]. Step 5 may be done in $O(M(t) \log t)$ using fast interpolation. See Ch 10 of [von zur Gathen and Gerhard 1999]. We summarize these complexity results in Table 2 below.

Step	Classical	Fast
2	$O(t^2)$	$O(M(t) \log t)$
3	$O(t^2 \log p)$	$O(M(t) \log t \log(pt))$
5	$O(t^2)$	$O(M(t) \log t)$

Table 2. Number of arithmetic operations in $\mathbb{Z}_p$ for t monomials.

1.3 Ben-Or/Tiwari with discrete logarithms

The discrete logarithm method modifies the Ben-Or/Tiwari algorithm so that the prime needed is a little larger than $(d+1)^n$ thus of size is $O(n \log d)$ bits instead of $O(D \log n \log \log n)$. [Murao and Fujise 1996] were the first to try this approach. Some practical aspects of it are discussed by van der Hoven and Lecerf in [van der Hoven and Lecerf 2014]. We explain how the method works.

To interpolate $C(x_1, \ldots, x_n)$ we first pick a prime p of the form $p = q_1 q_2 q_3 \ldots q_n + 1$ satisfying $2 | q_1$, $q_i > \deg_{x_i} C$ and $\gcd(q_i, q_j) = 1$ for $1 \leq i < j \leq n$. Finding such primes is not difficult and we omit presenting an explicit algorithm here.

Next we pick a random primitive element $\alpha \in \mathbb{Z}_p$ which we can do using the partial factorization $p - 1 = q_1 q_2 \ldots q_n$ (see [Stinson 2006]). We set $\omega_i = \alpha^{(p-1)/q_i}$ so that $\omega_i^{q_i} = 1$ and replace the evaluation points $(2^j, 3^j, \ldots, p_n^j)$ with $(\omega_1^j, \omega_2^j, \ldots, \omega_n^j)$. After Step 2 we factor $\lambda(z)$ in $\mathbb{Z}_p[z]$ to determine the m_i. If $M_i = \prod_{k=1}^{n} x_k^{d_k}$ we have $m_i = \prod_{k=1}^{n} \omega_k^{d_k}$. To compute d_k in Step 4 we compute the discrete logarithm $x := \log_\alpha m_i$, that is, we solve

$\alpha^x \equiv m_i \pmod{p}$ for $0 \le x < p-1$. We have

$$x = \log_\alpha m_i = \log_\alpha \prod_{k=1}^{n} \omega_k^{d_k} = \sum_{k=1}^{n} d_k \frac{p-1}{q_k}. \quad (3)$$

Taking (3) mod q_k we obtain $d_k = x[(p-1)/q_k]^{-1} \mod q_k$. Note the condition $\gcd(q_i, q_j) = 1$ ensures $(p-1)/q_k$ is invertible mod q_k. Step 5 remains unchanged.

For $p = q_1 q_2 \ldots q_n + 1$, a discrete logarithm can be computed in $O(\sum_{i=1}^{m} e_i(\log p + \sqrt{p}_i))$ multiplications in $\mathbb{Z}_p$ using the Pohlig-Helman algorithm where the factorization of $p-1 = \prod_{i=1}^{m} p_i^{e_i}$. See [Pohlig and Hellman 1978; Stinson 2006]. Since the $q_i \sim d$ this leads to an $O(n\sqrt{d})$ cost. Kaltofen showed in [Kaltofen et al. 2010] that this can be made polynomial in $\log d$ and n if one uses a Kronecker substitution to reduce multivariate interpolation to a univariate interpolation and uses a prime $p > (d+1)^n$ of the form $p = 2^k s + 1$ with s small.

1.4 Bad and Unlucky Evaluation Points

Let A and B be non constant polynomials in $\mathbb{Z}[x_0, \ldots, x_n]$, $G = \gcd(A, B)$ and $\bar{A} = A/G$ and $\bar{B} = B/G$. Let p be prime such that $LC(A)LC(B) \mod p \neq 0$.

Definition 1.6. Let $\alpha \in \mathbb{Z}_p^n$ and let $\bar{g}_\alpha(x) = \gcd(\bar{A}(x,\alpha), \bar{B}(x,\alpha))$. We say α is *bad* if $LC(A)(\alpha) = 0$ or $LC(B)(\alpha) = 0$ and α is *unlucky* if $\deg \bar{g}_\alpha(x) > 0$. If α is not bad and not unlucky we say α is *good.*

Example 1.7. Let $G = (x_1 - 16)x_0 + 1$, $\bar{A} = x_0^2 + 1$ and $\bar{B} = x_0^2 + (x_1 - 1)(x_2 - 9)x_0 + 1$. Then $LC(A) = LC(B) = x_1 - 16$ so $\{(16, \beta) : \beta \in \mathbb{Z}_p\}$ are bad and $\{(1, \beta) : \beta \in \mathbb{Z}_p\}$ and $\{(\beta, 9) : \beta \in \mathbb{Z}_p\}$ are unlucky.

Our GCD algorithm cannot reconstruct G using the image $g_\alpha(x) = \gcd(A(x,\alpha), B(x,\alpha))$ if α is unlucky. Brown's idea in [Brown 1971] to detect unlucky α is based on the following Lemma.

Lemma 1.8. *Let α and g_α be as above and $h_\alpha = G(x,\alpha) \mod p$. If α is not bad then $h_\alpha | g_\alpha$ and $\deg_x g_\alpha \ge \deg_x G$.*

For a proof of Lemma 1.8 see Lemma 7.3 of [Geddes et al. 1992]. Brown only uses α which are not bad and the images $g_\alpha(x)$ of least degree to interpolate G. The following Lemma implies if the prime p is large then unlucky evaluations points are rare.

Lemma 1.9. *If α is chosen at random from $\mathbb{Z}_p^n$ then*

$$\text{Prob}[\ \alpha \text{ is bad or unlucky }\ ] \le \frac{\deg AB + \deg A \deg B}{p}.$$

Proof: Let b be the number of bad evaluation points and let r be the number of unlucky evaluation points that are not also bad. Let B denote the event α is bad and G denote the event α is not bad and U denote the event α is unlucky. Then

$$\begin{aligned} \text{Prob}[B \text{ or } U] &= \text{Prob}[B] + \text{Prob}[G \text{ and } U] \\ &= \text{Prob}[B] + \text{Prob}[G] \times \text{Prob}[U|G] \\ &= \frac{b}{p^n} + \left(1 - \frac{b}{p^n}\right)\frac{r}{p^n - b} = \frac{b}{p^n} + \frac{r}{p^n}. \end{aligned}$$

Now α is bad $\Longrightarrow$ $\mathrm{LC}(A)(\alpha)\mathrm{LC}(B)(\alpha) = 0 \Longrightarrow LC(AB)(\alpha) = 0$. Applying Lemma 1.2 with $f = \mathrm{LC}(AB)$ we have $b \leq \deg LC(AB)p^{n-1}$. Let $R = \mathrm{res}_{x_0}(\bar{A}, \bar{B}) \in \mathbb{Z}_p[x_1, \ldots, x_n]$. Now α is unlucky and not bad $\Longrightarrow \deg \gcd(\bar{A}(x,\alpha), \bar{B}(x,\alpha)) > 0$ and $\mathrm{LC}(\bar{A})(\alpha) \neq 0$ and $\mathrm{LC}(\bar{B})(\alpha) \neq 0 \Longrightarrow R(\alpha) = 0$ by Lemma 1.5 (iv) and (v). Applying Lemma 1.2 we have $r \leq \deg(R)p^{n-1}$. Substituting into the above we have

$$\mathrm{Prob}[B \text{ or } U] \leq \frac{\deg \mathrm{LC}(AB)}{p} + \frac{\deg R}{p} \leq \frac{\deg AB}{p} + \frac{\deg A \deg B}{p} \quad \square$$

The following algorithm applies Lemma 1.9 to compute an upper bound d for $\deg_{x_i} G$.

Algorithm DegreeBound(A,B,i)

Input: Non-zero $A, B \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$ and i satisfying $0 \leq i \leq n$.
Output: $d \geq \deg_{x_i}(G)$ where $G = \gcd(A, B)$.

1 Set $LA = \mathrm{LC}(A, x_i)$ and $LB = \mathrm{LC}(B, x_i)$.
So $LA, LB \in \mathbb{Z}[x_0, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n]$.
2 Pick a prime $p \gg \deg A \deg B$ such that $LA \mod p \neq 0$ and $LB \mod p \neq 0$.
3 Pick $\alpha = (\alpha_0, \ldots, \alpha_{i-1}, \alpha_{i+1}, \ldots, \alpha_n) \in \mathbb{Z}_p^n$ at random until $LA(\alpha)LB(\alpha) \neq 0$.
4 Compute $a = A(\alpha_0, \ldots, \alpha_{i-1}, x_i, \alpha_{i+1}, \ldots, \alpha_n)$ and
$b = B(\alpha_0, \ldots, \alpha_{i-1}, x_i, \alpha_{i+1}, \ldots, \alpha_n)$.
5 Compute $g = \gcd(a, b)$ in $\mathbb{Z}_p[x_i]$ using the Euclidean algorithm.
6 Output $d = \deg_{x_i} g$.

1.5 Unlucky evaluations in Ben-Or/Tiwari

Consider again Example 1.7 where $G = (x_1 - 16)x_0 + 1$, $\bar{A} = x_0^2 + 1$ and $\bar{B} = x_0^2 + (x_1 - 1)(x_2 - 9)x_0 + 1$. For the Ben-Or/Tiwari points $\alpha_j = (2^j, 3^j)$ for $0 \leq j < 2t$ observe that $\alpha_0 = (1, 1)$ and $\alpha_2 = (4, 9)$ are unlucky and $\alpha_4 = (16, 81)$ is bad. Since none of these points can be used to interpolate G we need to modify the Ben-Or/Tiwari point sequence. For the GCD problem, we want random evaluation points to avoid bad and unlucky points. The following fix works.

Pick $0 < s < p$ at random and use $\alpha_j = (2^{s+j}, 3^{s+j}, \ldots, {p_n}^{s+j})$ for $0 \leq j < 2t$. Steps 1,2 and 3 work as before. To solve the *shifted* transposed Vandermonde system

$$W\,c = \begin{bmatrix} m_1^s & m_2^s & \ldots & m_t^s \\ {m_1}^{s+1} & {m_2}^{s+1} & \ldots & {m_t}^{s+1} \\ \vdots & \vdots & \vdots & \vdots \\ {m_1}^{s+t-1} & {m_2}^{s+t-1} & \ldots & m_t^{s+t-1} \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \\ \vdots \\ c_t \end{bmatrix} = \begin{bmatrix} v_s \\ v_{s+1} \\ \vdots \\ v_{s+t-1} \end{bmatrix} = u.$$

we first solve the transposed Vandermonde system

$$V\,b = \begin{bmatrix} 1 & 1 & \ldots & 1 \\ m_1 & m_2 & \ldots & m_t \\ \vdots & \vdots & \vdots & \vdots \\ {m_1}^{t-1} & {m_2}^{t-1} & \ldots & m_t^{t-1} \end{bmatrix} \begin{bmatrix} b_1 \\ b_2 \\ \vdots \\ b_t \end{bmatrix} = \begin{bmatrix} v_s \\ v_{s+1} \\ \vdots \\ v_{s+t-1} \end{bmatrix} = u$$

as before to obtain $b = V^{-1}u$. Observe that the matrix $W = VD$ where D is the t by t diagonal matrix with $D_{i,i} = m_i^s$. Solving $Wc = u$ for c we have

$$c = W^{-1}u = (VD)^{-1}u = (D^{-1}V^{-1})u = D^{-1}(V^{-1}u) = D^{-1}b.$$

Thus $c_i = b_i m_i^{-s}$ and we can solve $Wc = u$ in $O(t^2 + t\log s)$ multiplications.

Referring again to Example 2, if we use the discrete logarithm evaluation points $\alpha_j = (\omega_1^j, \omega_2^j)$ for $0 \le j < 2t$ then $\alpha_0 = (1,1)$ is unlucky and also, since $\omega_1^{q_1} = 1$, all $\alpha_{q_1}, \alpha_{2q_1}, \alpha_{3q_1}, \ldots$ are unlucky. Shifting the sequence to start at $j = 1$ and picking $q_i > 2t$ is problematic because for the GCD problem, t may be larger than $\max\{\#a_i, \#b_i\}$, or smaller; there is no way to know in advance. This difficulty led us to consider using a Kronecker substitution.

2 KRONECKER SUBSTITUTIONS

We propose to use a Kronecker substitution to map a multivariate polynomial GCD problem in $\mathbb{Z}[x_0, x_1, \ldots, x_n]$ into a bivariate GCD problem in $\mathbb{Z}[x, y]$. After making the Kronecker substitution, we need to interpolate $H(x,y) = \Delta(y)G(x,y)$ where $\deg_y H(x,y)$ will be exponential in n. To make discrete logarithms in $\mathbb{Z}_p$ feasible, we follow Kaltofen [Kaltofen et al. 2010] and pick $p = 2^k s + 1 > \deg_y H(x,y)$ with s small.

Definition 2.1. Let D be an integral domain and let f be a polynomial in $D[x_0, x_1, \ldots, x_n]$. Let $r \in \mathbb{Z}^{n-1}$ with $r_i > 0$. Let $K_r : D[x_0, x_1, \ldots, x_n] \to D[x, y]$ be the Kronecker substitution $K_r(f) = f(x, y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{n-1}})$.

Let $d_i = \deg_{x_i} f$ be the partial degrees of f for $1 \le i \le n$. We note that K_r is a homomorphism and it is invertible if $r_i > d_i$ for $1 \le i \le n-1$. Not all such Kronecker substitutions can be used, however, for the GCD problem. We consider an example.

Example 2.2. Consider the following GCD problem

$$G = x + y + z,\ \bar{A} = x^3 - yz,\ \bar{B} = x^2 - y^2$$

in $\mathbb{Z}[x,y,z]$. Since $\deg_y G = 1$ the Kronecker substitution $K_r(G) = G(x,y,y^2)$ is invertible. But $\gcd(K_r(\bar{A}), K_r(\bar{B})) = \gcd(\bar{A}(x,y,y^2), \bar{B}(x,y,y^2)) = \gcd(x^3 - y^3, x^2 - y^2) = x - y$. If we proceed to interpolate the $\gcd(K_r(A), K_r(B))$ we will obtain $(x - y)K_r(G)$ in expanded form from which and we cannot recover G.

We call such a Kronecker substitution unlucky. Theorem 2.5 below tells us that the number of unlucky Kronecker substitutions is finite. To detect them we will also avoid bad Kronecker substitutions in an analogous way Brown did to detect unlucky evaluation points.

Definition 2.3. Let K_r be a Kronecker substitution. We say K_r is *bad* if $\deg_x K_r(A) < \deg_{x_0} A$ or $\deg_x K_r(B) < \deg_{x_0} B$ and K_r is *unlucky* if $\deg_x \gcd(K_r(\bar{A}), K_r(\bar{B})) > 0$. If K_r is not bad and not unlucky we say K_r is *good.*

Proposition 2.4. *Let $f \in \mathbb{Z}[x_1, \ldots, x_n]$ be non-zero and $d_i \ge 0$ for $1 \le i \le n$. Let X be the number of Kronecker substitutions K_r such that $K_r(f) = 0$ where*

$$r \in \{[d_1 + k, d_2 + k, \ldots, d_{n-1} + k] \text{ for } k = 1, 2, 3, \ldots\}$$

Then $X \le (n-1)\sqrt{2\deg f}$.

PROOF: $K_r(f)=0 \iff f(y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{n-1}}) = 0$
$\iff f \bmod \langle x_1 - y, x_2 - y^{r_1}, \ldots, x_n - y^{r_1 r_2 \cdots r_{n-1}} \rangle = 0$
$\iff f \bmod \langle x_2 - x_1^{r_1}, x_3 - x_2^{r_2}, \ldots, x_n - x_{n-1}^{r_{n-1}} \rangle = 0$. Thus X is the number of ideals $I = \langle x_2 - x_1^{r_1}, \ldots, x_n - x_{n-1}^{r_{n-1}} \rangle$ for which $f \bmod I = 0$ with $r_i = d_i + 1, d_i + 2, \ldots$. We prove that $X \le (n-1)\sqrt{2\deg f}$ by induction on n.

If $n=1$ then I is empty so $f \bmod I = f$ and hence $X=0$ and the Lemma holds. For $n=2$ we have $f(x_1,x_2) \bmod \langle x_2 - x_1^{r_1}\rangle = 0 \implies x_2 - x_1^{r_1} | f$. Now X is maximal when $d_1 = 0$ and $r_1 = 1, 2, 3, \ldots$. We have

$$\textstyle\sum_{r_1=1}^{X} r_1 \le \deg f \implies X(X+1)/2 \le \deg f \implies X < \sqrt{2\deg f}.$$

For $n>2$ we proceed as follows. Either $x_n - x_{n-1}^{r_{n-1}} | f$ or it doesn't. If not then the polynomial $S = f(x_1, \ldots, x_{n-1}, x_{n-1}^{r_{n-1}})$ is non-zero. For the sub-case $x_n - x_{n-1}^{r_{n-1}} | f$ we obtain at most $\sqrt{2\deg f}$ such factors of f using the previous argument. For the case $S \ne 0$ we have

$$S \bmod I = 0 \iff S \bmod \langle x_2 - x_1^{r_1}, \ldots, x_{n-2} - x_{n-1}^{r_{n-2}} \rangle = 0$$

Notice that $\deg_{x_i} S = \deg_{x_i} f$ for $1 \le i \le n-2$. Hence, by induction on n, $X < (n-2)\sqrt{2\deg f}$ for this case. Adding the number of unlucky Kronecker substitutions for both cases yields $X \le (n-1)\sqrt{2\deg f}$. □

THEOREM 2.5. *Let* $A, B \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$ *be non-zero,* $G = \gcd(A,B)$, $\bar{A} = A/G$ *and* $B = \bar{B}/G$. *Let* $d_i \ge \deg_{x_i} G$. *Let* X *be the number of Kronecker substitutions* K_r *where* $r \in \{[d_1 + k, d_2 + k, \ldots, d_{n-1} + k]$ for $k = 1, 2, 3, \ldots\}$ *which are bad and unlucky. Then*

$$X \le \sqrt{2}(n-1)\left[\sqrt{\deg A} + \sqrt{\deg B} + \sqrt{\deg A \deg B}\right].$$

PROOF: Let $LA = \mathrm{LC}(A)$ and $LB = \mathrm{LC}(B)$ be the leading coefficients of A and B in x_0. Then K_r is bad $\iff K_r(LA) = 0$ or $K_r(LB) = 0$. Applying Proposition 2.4, the number of bad Kronecker substitutions is at most

$$(n-1)(\sqrt{2\deg LA} + \sqrt{2\deg LB}) \le (n-1)(\sqrt{2\deg A} + \sqrt{2\deg B}).$$

Now let $R = \mathrm{res}_{x_0}(\bar{A}, \bar{B})$. We will assume K_r is not bad.

$$\begin{aligned} K_r \text{ is unlucky} &\iff \deg_x(\gcd(K_r(\bar{A}), K_r(\bar{B})) > 0 \\ &\iff \mathrm{res}_x(K_r(\bar{A}), K_r(\bar{B})) = 0 \\ (K_r \text{ is not bad and is a homomorphism}) &\iff K_r(\mathrm{res}_x(\bar{A}, \bar{B})) = 0 \\ &\iff K_r(R) = 0 \end{aligned}$$

By Proposition 2.4, the number of unlucky Kronecker substitutions $\le (n-1)\sqrt{2\deg R} \le (n-1)\sqrt{2\deg A \deg B}$ by Lemma 1.5(ii). Adding the two contributions proves the theorem. □

Theorem 2.5 tells us that the number of unlucky Kronecker substitutions is finite. Our algorithm, after identifying an unlucky Kronecker substitution will try the next Kronecker substitution $r = [r_1 + 1, r_2 + 1, \ldots, r_{n-1} + 1]$.

It is still not obvious that a Kronecker substitution that is not unlucky can be used because it can create a content in y of exponential degree. The following example shows how we recover $H = \Delta G$ when this happens.

Example 2.6. Consider the following GCD problem

$$G = wx^2 + zy,\ \bar{A} = ywx + z,\ \bar{B} = yzx + w$$

in $\mathbb{Z}[x, y, z, w]$. We have $\Gamma = wy$ and $\Delta = y$. For $K_r(f) = f(x, y, y^3, y^9)$ we have

$$\gcd(K_r(A), K_r(B)) = K_r(G)\gcd(y^{10}x + y^3, y^4x + y^9) = (y^9x^2 + y^4)y^3 = y^7(y^5x^2 + 1).$$

One must not try to compute $\gcd(K_r(A), K_r(B))$ because the degree of the content of $\gcd(K_r(A), K_r(B))$ (y^7 in our example) can be exponential in n the number of variables and we cannot compute this efficiently using the Euclidean algorithm. The crucial observation is that if we compute **monic** images $g_j = \gcd(K_r(A)(x, \alpha^j), K_r(B)(x, \alpha^j))$ any content is divided out, and when we scale by $K_r(\Gamma)(\alpha^j)$ and interpolate y in $K_r(H)$ using sparse interpolation, we recover any content. We obtain $K_r(H) = K_r(\Delta)K_r(G) = y^{10}x^2 + y^5$, then invert K_r to obtain $H = (yw)x^2 + (y^2z)$.

2.1 Unlucky primes

Let A, B be polynomials in $\mathbb{Z}[x_0, x_1, \ldots, x_n]$, $G = \gcd(A, B)$, $\bar{A} = A/G$ and $\bar{B} = B/G$. In the introduction we defined the polynomials $\Gamma = \gcd(LC(A), LC(B)$, $\Delta = \Gamma/LC(G)$ and $H = \Delta G$ where $LC(A)$, $LC(B)$ and $LC(G)$ are the leading coefficients of A, B and G in x_0 respectively.

Let $K_r : \mathbb{Z}[x_0, x_1, \ldots, x_n] \to \mathbb{Z}[x, y]$ be a Kronecker substitution $K_r(f) = f(x, y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{n-1}})$ for some $r_i > 0$. Our GCD algorithm will compute $\gcd(K_r(A), K_r(B))$ modulo a prime p. Some primes cannot be used.

Example 2.7. Consider the following GCD problem in $\mathbb{Z}[x_0, x_1]$ where a and b are positive integers.

$$G = x_0 + b\,x_1 + 1,\ \bar{A} = x_0 + x_1 + a,\ \bar{B} = x_0 + x_1$$

In this example, $\Gamma = 1$ so $H = G$. Since there are only two variables the Kronecker substitution is $K_r(f) = f(x, y)$ hence $K_r(\bar{A}) = x + y + a$, $K_r(\bar{B}) = x + y$. Notice that $\gcd(K_r(\bar{A}), K_r(\bar{B})) = 1$ in $\mathbb{Z}[x, y]$, but $\gcd(\phi_p(K_r(\bar{A})), \phi_p(K_r(\bar{B}))) = x + y$ for any prime $p|a$. Like Brown's modular GCD algorithm in [Brown 1971], our GCD algorithm must avoid these primes.

If our GCD algorithm were to choose primes from a pre-computed set of primes $S = \{p_1, p_2, \ldots, p_N\}$ then notice that if we replace a in example 2.7 with $a = \Pi_{i=1}^{N} p_i$ then every prime would be unlucky. To guarantee that our GCD algorithm will succeed on all inputs we need to bound the number of primes that cannot be used and pick our prime from a sufficiently large set at random.

Because our algorithm will always choose $r_i > \deg_{x_i} H$, the Kronecker substitution K_r leaves the coefficients of H unchanged. Let $pmin$ be the smallest prime in S. From Section

1, $H = \sum_{i=0}^{dG} h_i x_0^i$ with $t = \max(\#h_i)$, we have $\#H \le (d+1)t$ hence if p is chosen at random from S then

$$\text{Prob}[\text{Supp}(\phi_p(H)) \ne \text{Supp}(H)] \le \frac{(d+1)t \log_{pmin} \|H\|}{N}.$$

Theorem 2.10 below bounds $\|H\|$ from the inputs A and B.

Definition 2.8. Let p be a prime and let K_r be a Kronecker substitution. We say p is *bad* if $\deg_x \phi_p(K_r(A)) < \deg_x K_r(A)$ or $\deg_x \phi_p(K_r(B)) < \deg_x K_r(B)$ and p is *unlucky* if $\deg_x \gcd(\phi_p(\bar{K}_r(A)), \phi_p(\bar{K}_r(B))) > 0$. If p is not bad and not unlucky we say p is *good.*

Let $R = \text{res}_x(\bar{A}, \bar{B}) \in \mathbb{Z}[x_1, \ldots, x_n]$ be the Sylvester resultant of $\bar{A}$ and $\bar{B}$. Unlucky primes are characterized as follows; if p is not bad then Lemma 1.5(vii) implies p is unlucky $\iff$ $\phi_p(K_r(R)) = 0$. Unlucky primes are detected using the same approach as described for unlucky evaluations in section 1.3 which requires that we also avoid bad primes. If p is bad or unlucky then p must divide the integer $M = \|K_r(LC(A))\| \cdot \|K_r(LC(B))\| \cdot \|K_r(R)\|$. Let $pmin = \min_{i=1}^{N} p_i$. Thus if p is chosen at random from S then

$$\text{Prob}[\ p \text{ is bad or unlucky }] \le \frac{\log_{pmin} M}{N}.$$

Proposition 2.9. *Let A be an $m \times m$ matrix with entries $A_{i,j} \in \mathbb{Z}[x_1, x_2, \ldots, x_n]$ satisfying the term bound $\#A_{i,j} \le t$, the degree bound $\deg_{x_k} A_{i,j} \le d$ and the coefficient bound $\|A_{i,j}\| < h$ (for $1 \le i,j \le m$). Note if a term bound for $\#A_{i,j}$ is not known we may use $t = (1+d)^n$. Let $K_r : \mathbb{Z}[x_1, x_2, \ldots, x_n] \to \mathbb{Z}[y]$ be the Kronecker map $K_r(f) = f(y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{n-1}})_k > 0$ and let $B = K_r(A)$ be the $m \times m$ matrix of polynomials in $\mathbb{Z}[y]$ with $B_{i,j} = K_r(A_{i,j})$ for $1 \le i,j \le m$. Then*

(i) $\|\det A\| < m^{m/2} t^m h^m$ *and*
(ii) $\|\det B\| < m^{m/2} t^m h^m$.

proof: To prove (i) let S be the $m \times m$ matrix of integers given by $S_{i,j} = \|A_{i,j}\|_1$ We claim $\|\det A\| \le H(S)$ where $H(S)$ is Hadamard's bound on $|\det S|$. Then applying Hadamard's bound to S we have

$$H(S) = \prod_{i=1}^{m} \sqrt{\sum_{j=1}^{m} S_{i,j}^2} = \prod_{i=1}^{m} \sqrt{\sum_{j=1}^{m} \|A_{i,j}\|_1^2} < \prod_{i=1}^{m} \sqrt{m(th)^2} = m^{m/2} t^m h^m$$

which establishes (i).

To prove our claim let K_s be a Kronecker map with $s_i > md$ and let C be the $m \times m$ matrix with $C_{i,j} = K_s(A_{i,j})$. Notice that $\deg_{x_k} A_{i,j} \le d$ implies $\deg_{x_k} \det A \le md$ for $1 \le k \le n$. Thus $K_s(\det A)$ is a bijective map on the monomials of $\det A$ thus $K_s(\det A) = \det C$ which implies $\|\det A\| = \|\det C\|$. Now let W be the $m \times m$ matrix with $W_{i,j} = \|C_{i,j}\|_1$ and let $H(W)$ be Hadamard's bound on $|\det W|$. Then $\|\det C\| \le H(W)$ by Lemma 1.4 and since K_s is bijective $S = W$ hence $H(S) = H(W)$. Therefore $\|\det A\| = \|\det C\| \le H(W) = H(S)$ which proves the claim.

To prove (ii), let S and T be the $m \times m$ matrices of integers given by $S_{i,j} = \|A_{i,j}\|_1$ and $T_{i,j} = \|B_{i,j}\|_1$ for $1 \le i,j \le m$. From the claim in part (i) if $r_k > md$ we have

$\|\det A\| = \|\det B\| \le H(T) = H(S)$. Now if $r_k \le md$ for any $1 \le k \le n-1$ then $K_r(\det A)$ is not necessarily one-to-one on the monomials in $\det A$. However, for all $r_k > 0$ we still have

$$\|K_r(A_{i,j})\|_1 \;\le\; \|A_{i,j}\|_1 \le i, j \le m$$

so that $T_{i,j} \le S_{i,j}$ hence $H(T) \le H(S)$. We have $\|\det B\| \le H(T) \le H(S)$ and (ii) follows. □

Theorem 2.10. *Let $A, B, G, \bar{A}, \bar{B}, \Delta, H$ be as given at the beginning of this section and let $R = \mathrm{res}_{x_0}(\bar{A}, \bar{B})$. Suppose $A = \sum_{i=0}^{d_A} a_i(x_1, \ldots, x_n)x_0^i$ and $B = \sum_{i=0}^{d_B} b_i(x_1, \ldots, x_n)x_0^i$ satisfy $\deg A \le d$, $\deg B \le d$, $d_A > 0$, $d_B > 0$, $\|a_i\| < h$ and $\|b_i\| < h$. Let $K_r : \mathbb{Z}[x_0, x_1, \ldots, x_n] \to \mathbb{Z}[x, y]$ be the Kronecker map $K_r(f) = f(x, y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{n-1}})$. If K_r is not bad, that is, $K_r(a_{dA}) \neq 0$ and $K_r(a_{dB}) \neq 0$, then*

(i) $\|K_r(LC(A))\| \le (1+d)^n h$ and $\|K_r(LC(B))\| \le (1+d)^n h$,
(ii) $\|K_r(R)\| \le m^{m/2}(1+d)^{nm}E^m$ and
(iii) if $r_i > \deg_{x_i} H$ for $1 \le i \le n-1$ then $\|H\| \le (1+d)^n E^2$

where $m = d_A + d_B$ and $E = \mathrm{e}^{(n+1)d}h$.

Proof: Since $LC(A) \in \mathbb{Z}[x_1, \ldots, x_n]$ we have $\#LC(A) \le (1+d)^n$ thus $\|K_r(LC(A))\| \le (1+d)^n\|LC(A)\| \le (1+d)^n h$. Using the same argument we have $\|K_r(LC(B))\| \le (1+d)^n h$ which proves (i).

Let $\bar{A} = \sum_{i=0}^{d\bar{A}} \bar{a}_i x_0^i$ and $\bar{B} = \sum_{i=0}^{d\bar{B}} \bar{b}_i x_0^i$. Because $A = G\bar{A}$ and $B = G\bar{B}$, Lemma 1.3 implies $\|\bar{A}\| < E$ and $\|\bar{B}\| < E$. Let S be Sylvester's matrix formed from $K_r(\bar{a}_i)$ and $K_r(\bar{b}_i)$. Now $K_r(R) = \det S$ and S has dimension $d\bar{A} + d\bar{B} \le dA + dB = m$. Applying Proposition 2.9 to S we have

$$\|K_r(R)\| = \|\det S\| \le t^m E^m m^{m/2}$$

where $t = \max_{i,j} \#S_{i,j}$. Since $\bar{A}|A$ and $\bar{B}|B$ we have $\deg_{x_j} \bar{a}_i(x_1, \ldots, x_n) \le d$ and $\deg_{x_j} \bar{b}_i(x_1, \ldots, x_n) \le d$ thus $\#S_{i,j} \le (1+d)^n$ and (ii) follows.

For (iii) since $G|A$ and $\Delta|LC(A)$, Lemma 1.3 implies $\|G\| < E$ and $\|\Delta\| < E$. Thus

$$\|H\| = \|\Delta G\| \le \#\Delta \cdot \|\Delta\| \cdot \|G\| \le (1+d)^n E^2. \;\square$$

We remark that our definition for unlucky primes differs from Brown [Brown 1971]. Brown's definition depends on the vector degree whereas ours depends only on the degree in x_0 the main variable. The following example illustrates the difference.

Example 2.11. Consider the following GCD problem and prime p.

$$G = x + y + 1,\;\; \bar{A} = (y+p)x^2 + y^2,\;\; \bar{B} = yx^3 + y + p.$$

We have $\gcd(\phi_p(\bar{A}), \phi_p(\bar{B})) = \gcd(yx + y^2, yx^2 + y) = y$. By definition 2.8, p is not unlucky but by Brown's definition, p is unlucky.

Our GCD algorithm in $\mathbb{Z}[x_0, x_1, \ldots, x_n]$ only needs monic images in $\mathbb{Z}_p[x_0]$ to recover H whereas Brown needs monic images in $\mathbb{Z}_p[x_0, x_1, \ldots, x_n]$ to recover G. A consequence of this is that our bound on the number of unlucky primes is much smaller than Brown's bound (see Theorems 1 and 2 of [Brown 1971]). This is relevant because we also require p to be smooth.

2.2 The number of unlucky evaluation points

Even if the Kronecker substitution is not unlucky, after applying it to input polynomials A and B, because the degree in y may be very large, the number of bad and unlucky evaluation points may be very large.

Example 2.12. Consider the following GCD problem

$G = x_0 + x_1^d + x_2^d + \cdots + x_n^d$, $\bar{A} = x_0 + x_1 + \cdots + x_{n-1} + x_n^{d+1}$, $\bar{B} = x_0 + x_1 + \cdots + x_{n-1} + 1$.

To recover G, if we use $r = [d+1, d+1, \ldots, d+1]$ for $x_1, x_2, \ldots, x_{n-1}$ we need $p > (d+1)^n$. But $R = \mathrm{res}_{x_0}(\bar{A}, \bar{B}) = 1 - x_n^{d+1}$ and $K_r(R) = 1 - (y^{r_1 r_2 \cdots r_{n-1}})^{d+1} = 1 - y^{(d+1)^n}$ which means there could be as many as $(d+1)^n$ unlucky evaluation points. If $p = (d+1)^n + 1$, all evaluation points would be unlucky.

To guarantee that we avoid unlucky evaluation points with high probability we would need to pick $p \gg \deg K_r(R)$ which could be much larger than what is needed to interpolate $K_r(H)$. But this upper bound based on the resultant is a worst case. This lead us to investigate what the expected number of unlucky evaluation points is. We ran an experiment. We computed all monic quadratic and cubic bivariate polynomials over small finite fields $\mathbb{F}_q$ of size $q = 2, 3, 4, 5, 7, 8, 11$ and counted the number of unlucky evaluation points to find the following result.

Theorem 2.13. *Let $\mathbb{F}_q$ be a finite field with q elements and $f = x^l + \sum_{i=0}^{l-1}(\sum_{j=0}^{d_i} a_{ij} y^j) x^i$ and $g = x^m + \sum_{i=0}^{m-1}(\sum_{j=0}^{e_i} b_{ij} y^j) x^i$ with $l \geq 1$, $m \geq 1$, and $a_{ij}, b_{ij} \in \mathbb{F}_q$. Let $X = |\{\alpha \in \mathbb{F}_q : \gcd(f(x,\alpha), g(x,\alpha)) \neq 1\}|$ be a random variable over all choices $a_{ij}, b_{ij} \in \mathbb{F}_q$. So $0 \leq X \leq q$ and for f and g not coprime in $\mathbb{F}_q[x,y]$ we have $X = q$. If $d_i \geq 0$ and $e_i \geq 0$ then $\mathrm{E}[X] = 1$.*

proof: Let $C(y) = \sum_{i=0}^{d} c_i y^i$ with $d \geq 0$ and $c_i \in \mathbb{F}_q$ and fix $\beta \in \mathbb{F}_q$. Consider the evaluation map $C_\beta : \mathbb{F}_q^{d+1} \to \mathbb{F}_q$ given by $C_\beta(c_0, \ldots, c_d) = \sum_{i=0}^{d} c_i \beta^i$. We claim that C is balanced, that is, C maps q^d inputs to each element of $\mathbb{F}_q$. It follows that $f(x,\beta)$ is also balanced, that is, over all choices for $a_{i,j}$ each monic polynomial in $\mathbb{F}_q[x]$ of degree n is obtained equally often. Similarly for $g(x,\beta)$.

Recall that two univariate polynomials a, b in $\mathbb{F}_q[x]$ with degree $\deg a > 0$ and $\deg b > 0$ are coprime with probability $1 - 1/q$ (see Ch 11 of Mullen and Panario [Mullen and Panario 2013]). This is also true under the restriction that they are monic. Therefore $f(x,\beta)$ and $g(x,\beta)$ are coprime with probability $1 - 1/q$. Since we have q choices for β we obtain

$$E[X] = \sum_{\beta \in \mathbb{F}_q} \mathrm{Prob}[\gcd(A(x,\beta), B(x,\beta)) \neq 1] = q(1 - (1 - \frac{1}{q})) = 1.$$

Proof of claim. Since $B = \{1, y-\beta, (y-\beta)^2, \ldots, (y-\beta)^d\}$ is a basis for polynomials of degree d we can write each $C(y) = \sum_{i=0}^{d} c_i y^i$ as $C(y) = u_0 + \sum_{i=1}^{d} u_i (y-\beta)^i$ for a unique choice of $u_0, u_1, \ldots, u_d \in \mathbb{F}_q$. Since $C(\beta) = u_0$ it follows that all q^d choices for $u_1, \ldots, u_d$ result in $C(\beta) = u_0$ hence C is balanced. □

That $E[X] = 1$ was a surprise to us. We thought $E[X]$ would have a logarithmic dependence on $\deg f$ and $\deg g$. In light of Theorem 2.13, we will first pick $p > \deg_y(K_r(H))$

and, should the algorithm encounter unlucky evaluations, restart the algorithm with a larger prime.

3 SIMPLIFIED ALGORITHM

We now present our GCD algorithm. It consists of two parts: the main routine MGCD and the subroutine PGCD. PGCD computes the GCD modulo a prime and MGCD calls PGCD several times to obtain enough images to reconstruct the coefficients of the target polynomial H using Chinese Remaindering. In this section, we assume that we are given a term bound τ on the number of terms in the coefficients of target polynomial H, that is $\tau \geq \#h_i(x_1, x_2, \ldots, x_n)$. We will also choose a Kronecker substitution that is a priori not bad and not unlucky. These assumptions will enable us to choose the prime p so that PGCD computes G with high probability. We will relax these assumptions in the next section. The algorithm will need to treat bad and unlucky primes and bad and unlucky evaluation points.

3.1 Bad and unlucky Kronecker substitutions

Lemma 3.1. *Let $K_r : \mathbb{Z}[x_0, x_1, \ldots, x_n] \to \mathbb{Z}[x, y]$ be the Kronecker substitution $K_r(f) := f(x, y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{n-1}})$. If $f \neq 0$ and $r_i > \deg_{x_i}(f)$ for $1 \leq i \leq n-1$ then $K_r(f)$ sends monomials in f to unique monomials and therefore K_r is one-to-one and $K_r(f) \neq 0$.*

Proof. Suppose two monomials $x_0^{d_0} x_1^{d_1} \cdots x_n^{d_n}$ and $x_0^{e_0} e_1^{e_1} \cdots x_n^{e_n}$ in f are mapped to the same monomial in $\mathbb{Z}[x, y]$ so that

$$x^{d_0} y^{d_1} y^{r_1 d_2} \cdots y^{r_1 r_2 \cdots r_{n-1} d_n} = x^{e_0} y^{e_1} y^{r_1 e_2} \cdots y^{r_1 r_2 \cdots r_{n-1} e_n}$$

Clearly $d_0 = e_0$ and

$$d_1 + r_1 d_2 + \cdots + r_1 r_2 \cdots r_{n-1} d_n = e_1 + r_1 e_2 + \cdots + r_1 r_2 \cdots r_{n-1} e_n \tag{4}$$

Reducing (4) modulo r_1 we have $d_1 \equiv e_1 \pmod{r_1}$. Now $r_1 > \deg_{x_1} f$ implies $r_1 > d_1$ and $r_1 > e_1$ implies $d_1 = e_1$. Subtracting $d_1 = e_1$ from this equation and dividing through by r_1 we have

$$d_2 + r_2 d_3 + \ldots r_2 r_3 \cdots r_{n-1} d_n = e_2 + r_2 e_3 + \ldots r_2 r_3 \cdots r_{n-1} e_n$$

Repeating the argument we obtain $d_i = e_i$ for $1 \leq i \leq n$. □

In our case, we are considering the polynomials $A, B \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$ with $\deg_{x_0} A > 0$ and $\deg_{x_0} B > 0$. Let $G = \gcd(A, B)$ and $\bar{A} = A/G$ and $\bar{B} = B/G$ and let $LC(A)$ and $LC(B)$ the the leading coefficients of A and B with respect to x_0. Lemma 3.1 implies that if we pick $r_i > \max(\deg_{x_i} LC(A), \deg_{x_i} LC(B))$ then $K_r(LC(A)) \neq 0$ and $K_r(LC(B)) \neq 0$ thus K_r is not bad. Let $R = \mathrm{res}_{x_0}(\bar{A}, \bar{B})$. By Lemma 1.5(iii), we have

$$\deg_{x_i} R \leq \deg x_0 \bar{B} \deg_{x_i} \bar{A} + \deg x_0 \bar{A} \deg_{x_i} \bar{B}.$$

Since $\deg_{x_i} \bar{A} \leq \deg_{x_i} A$ and $\deg_{x_i} \bar{B} \leq \deg_{x_i} B$ for $0 \leq i \leq n$ we have

$$\deg_{x_i} R \leq \deg_{x_0} B \deg_{x_i} A + \deg_{x_0} A \deg_{x_i} B.$$

So if we pick $r_i = (\deg_{x_0} B \deg_{x_i} A + \deg_{x_0} A \deg_{x_i} B) + 1$, then K_r is always lucky by Lemma 3.1. The assumption that $\deg_{x_0} A > 0$ and $\deg_{x_0} B > 0$ gives

$$(\deg_{x_0} B \deg_{x_i} A + \deg_{x_0} A \deg_{x_i} B) \geq \max\{\deg_{x_i} LC(A), \deg_{x_i} LC(B))\}$$

hence the Kronecker substitution K_r with the sequence

$$[r_i = (\deg_{x_i} A \deg_{x_0} B + \deg_{x_i} B \deg_{x_0} A) + 1]_{1 \leq i \leq n}$$

is good.

3.2 Bad and unlucky evaluations

In this section, the Kronecker substitution K_r is assumed to be good. We also assume that the prime p is good.

Proposition 3.2. *Let $d = \max\{\max\{\deg_{x_i} A, \deg_{x_i} B\}_{0 \leq i \leq n}\}$ and let $r_i = 2d^2 + 1$ for $1 \leq i \leq n$. Note $2d^2 + 1 \geq (\deg_{x_i} A \deg_{x_0} B + \deg_{x_i} B \deg_{x_0} A) + 1$. Then*

(1) $\deg_y K_r(A) < (2d^2+1)^n$ and $\deg_y K_r(B) < (2d^2+1)^n$,
(2) $\deg_y LC(K_r(A))(y) < (2d^2+1)^n$ and $\deg_y LC(K_r(B))(y) < (2d^2+1)^n$,
(3) $\deg_y K_r(H) < (2d^2+1)^n$, and
(4) $\deg_y K_r(R) < 2d(2d^2+1)^n$, where $K_r(R) = \mathrm{res}_x(K_r(\bar{A}), K_r(\bar{B}))$.

Proof. For (1), after the Kronecker substitution, the exponent of $y \leq e_1 + e_2(2d^2+1) + \cdots + e_n(2d^2+1)^{n-1}$, where e_i is the exponent of x_i and $e_i \leq d$ for all i. So $\deg_y K_r(A)$ and $\deg_y K_r(B)$ are bounded by

$$\begin{aligned}
d + d(2d^2+1) + \cdots + d(2d^2+1)^{n-1} &= d(1 + (2d^2+1) + \cdots + (2d^2+1)^{n-1}) \\
&= d(1 + \frac{(2d^2+1)^n - (2d^2+1)}{(2d^2+1) - 1}) \\
&= \frac{2d^3}{2d^2} + \frac{d(2d^2+1)^n - d(2d^2+1)}{2d^2}) \\
&= \frac{d(2d^2+1)^n - d}{2d^2} \\
&< (2d^2+1)^n.
\end{aligned}$$

Property (2) follows from (1). For (3), recall that $\deg_y K_r(H) = \deg_y K_r(\Delta G)$. Since $\Delta = \gcd(\mathrm{LC}(\bar{A}), LC(\bar{B}))$, we have

$$\begin{aligned}
\deg_y K_r(\Delta G) &= \deg_y K_r(\Delta) + \deg_y K_r(G) \\
&\leq \min(\deg_y K_r(LC(\bar{A})), \deg_y K_r(LC(\bar{B}))) + \deg_r K_r(G) \\
&\leq \min(\deg_y K_r(\bar{A}), \deg_y K_r(\bar{B})) + \deg_r K_r(G) \\
&= \min(\deg_y K_r(A), \deg_y K_r(B)) < (2d^2+1)^n.
\end{aligned}$$

For (4),

$$\deg_y K_r(R) \leq \deg_y K_r(\bar{A}) \deg_x K_r(\bar{B}) + \deg_y K_r(\bar{B}) \deg_x K_r(\bar{A}),$$

where $\deg_x K_r(\bar{A}) = \deg_{x_0} \bar{A} \leq \deg_{x_0} A \leq d$, $\deg_x K_r(\bar{B}) = \deg_{x_0} \bar{B} \leq \deg_{x_0} B, \leq d$, and $\deg_y K_r(\bar{A}) \leq \deg_y K_r(A)$ and $\deg_y K_r(\bar{B}) \leq \deg_y K_r(B)$. So we have

$$\deg_y K_r(R) < d(2d^2+1)^n + d(2d^2+1)^n = 2d(2d^2+1)^n.$$

□

By proposition 3.2(1), a prime $p > (2d^2+1)^n$ is sufficient to recover the exponents for the Kronecker substitution. With the assumption that p is not bad and not unlucky, we have the following lemma.

Lemma 3.3. *Let p be a prime. If α is chosen at random from $[0, p-1]$, then*

(i) $\text{Prob}[\alpha \textit{ is bad}] < \dfrac{2(2d^2+1)^n}{p}$ *and*

(ii) $\text{Prob}[\alpha \textit{ is unlucky or } \alpha \textit{ is bad}] < \dfrac{(2d+2)(2d^2+1)^n}{p}$.

Proof. $\text{Prob}[\alpha \text{ is bad}] = \text{Prob}[LC(K_r(A)(\alpha)\, LC(K_r(B))(\alpha) = 0]$ $\leq \deg LC(K_r(AB))(y)/p < 2(2d^2+1)^n/p$. For (ii) from the proof of Lemma 1.9 we have this probability $\leq \deg K_r(\text{LC}(AB))/p + \deg K_r(R)/p$ where $R = \text{res}_{x_0}(\bar{A}, \bar{B})$. Applying proposition 3.2(1) and (4) we have the probability $< 2(2d^2+1)^n/p + 2d(2d^2+1)^n/p$ and the result follows. □

The probability that our algorithm does not encounter a bad or unlucky evaluation can be estimated as follows. Let U denote the bound of the number of bad and unlucky evaluation points and $\tau \geq max_i\{\#h_i\}$. We need 2τ good consecutive evaluation points (a segment of length 2τ in the sequence $(1, \ldots, p-1)$) to compute the feedback polynomial for h_i. Suppose α^k is a bad or unlucky evaluation point where $s \leq k < s + 2\tau - 1$ for any positive integer $s \in (0, p-1]$. Then every segment of length 2τ starting at α^i where $k - 2\tau + 1 \leq i \leq k$ includes the point α^k. Hence our algorithm fails to determine the correct feedback polynomial. The union of all segments including α^k has length $4\tau - 1$. We can not use every segment of length 2τ from $k - 2\tau + 1$ to $k + 2\tau - 1$ to construct the correct feedback polynomial. The worst case occurs when all bad and unlucky evaluation points, their corresponding segments of length $4\tau - 1$ do not overlap. Since there are at most U of them, we can not determine the correct feedback polynomials for at most $U(4\tau - 1)$ points. Note, this does not mean that all those points are bad or unlucky, there is only one bad or unlucky point in each segment of length 2τ. U is bounded by $2(2d^2+1)^n + 2d(2d^2+1)^n = (2d+2)(2d^2+1)^n$.

Lemma 3.4. *Suppose p is good. Then*

$$\text{Prob}[2\tau \textit{ evaluation points fail to determine the feedback polynomial}]$$
$$\leq \frac{4\tau U - U}{p-1} < \frac{4\tau U}{p-1} = \frac{4\tau(2d+2)(2d^2+1)^n}{p-1}.$$

So if we choose a prime $p > 4X\tau(2d+2)(2d^2+1)^n$ for some positive number X, then the probability that PGCD fails is at most $\frac{1}{X}$.

We note that the choice of p in previous lemma implies $p > (2d^2+1)^n \geq \deg_y(K_r(H))$. So we can recover the exponents of y in H.

3.3 Bad and unlucky primes

Our goal here is to construct a set S of smooth primes, with $|S|$ large enough so if we choose a prime $p \in S$ at random, the probability that p is good is at least $\frac{1}{2}$. Recall that a prime p is said to be y-smooth if $q|p-1$ implies $q \leq y$. The choice of y affects the efficiency of discrete logarithm computation in $\mathbb{Z}_p$.

A bad prime must divide $\|LC(K_r(A))\|$ or $\|LC(K_r(B))\|$ and an unlucky prime must divide $\|Kr(R)\|$. Recall that in section 2.1,

$$M = \|LC(K_r(A))\|\|LC(K_r(B))\|\|Kr(R)\|.$$

We want to construct a set $S = \{p_1, p_2, \dots, p_N\}$ of N smooth primes with each $p_i > 4\tau(2d+4)(2d^2+1)^n X$. If $p > 4\tau(2d+4)(2d^2+1)^n X$, then the probability that our algorithm fails to determine the feedback polynomial is $< \frac{1}{X}$. The size N of S can be estimated as follows. If

$$N = Y\lceil \log_{4X\tau(2d+4)(2d^2+1)^n} M\rceil > Y \log_{pmin} M,$$

where a bound for M is given by Theorem 2.10 (ii), $pmin = \min_{p_i \in S} p_i$ and $Y > 0$, Then

$$\text{Prob}[\ p \text{ is bad or unlucky}\ ] \leq \frac{\log_{pmin} M}{N} < \frac{1}{Y}.$$

We construct the set S which consists of N y-smooth primes so that $\min_{p_i \in S} p_i > 4\tau X(2d+4)(2d^2+1)^n$ which is the constraint for the bad or unlucky evaluation case. We conclude the following result.

Theorem 3.5. *Let S be constructed as just described. Let p be chosen at random from S, s be chosen at random from $0 < s \leq p-1$ and α_p be a random generator of $\mathbb{Z}_p^*$. Let $E = \{\alpha_p^{s+j} : 0 \leq j < 2\tau\}$ be 2τ consecutive evaluation points. For any $X > 0$ and $Y > 0$, we have*

$$\text{Prob}[p \text{ is good and } E \text{ are all good}] > (1 - \frac{1}{X})(1 - \frac{1}{Y}).$$

3.4 The Simplified GCD Algorithm

Let $S = \{p_1, p_2, \dots, p_N\}$ is the set of N primes constructed in the previous section. We've split our GCD algorithm into two subroutines, subroutine MGCD and PGCD. The main routine MGCD chooses a Kronecker substitution K_r and then chooses a prime p from S at random and calls PGCD to compute $K_r(H) \bmod p$.

Algorithm MGCD is a Las Vegas algorithm. The choice of S means that algorithm PGCD will compute $K_r(H) \mod p$ with probability at least $(1-\frac{1}{X})(1-\frac{1}{Y})$. By taking $X = 4$ and $Y = 4$ this probability is at least $\frac{1}{2}$. The design of MGCD means that even with probability $\frac{1}{2}$, the expected number of calls to algorithm PGCD is linear in the minimum number of primes needed to recover H using Chinese remaindering, that is, we do not need to make the probability that algorithm PGCD computes $H \mod p$ high for algorithm MGCD to be efficient.

Algorithm MGCD(A, B, τ)

Inputs $A, B \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$ and a term bound τ satisfying $n > 0$, A and B are primitive in x_0, $\deg_{x_0} A > 0$, $\deg_{x_0} B > 0$ and $\tau \geq \max \#h_i$.

Output $G = \gcd(A, B)$.

1 Compute $\Gamma = \gcd(LC(A), LC(B))$ in $\mathbb{Z}[x_1, \ldots, x_n]$.
2 Set $r_i = 1 + (\deg_{x_i} A \deg_{x_0} B + \deg_{x_i} B \deg_{x_0} A)$ for $1 \leq i < n$.
3 Let $Y = (y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{n-1}})$.
Set $K_r A = A(x, Y)$, $K_r B = B(x, Y)$ and $K_r \Gamma = \Gamma(Y)$.
4 Construct the set S of smooth primes according to Theorem 3.5 with $X = 4$ and $Y = 4$.
5 Set $\widehat{H} = 0, M = 1, d_0 = \min(\deg_{x_0} A, \deg_{x_0} B)$.

LOOP: // Invariant: $d_0 \geq \deg_{x_0} H = \deg_{x_0} G$.

6 Call PGCD($K_r A$, $K_r B$, $K_r \Gamma$, S, τ, M).
If PGCD outputs FAIL then **goto** LOOP.
Let p and $\widehat{H}p = \sum_{i=0}^{dx} \widehat{h}_i(y) x^i$ be the output of PGCD.
7 If $dx > d_0$ then either p is unlucky or all evaluation points were unlucky so **goto** LOOP.
8 If $dx < d_0$ then either this is the first image or all previous images in $\widehat{H}$ were unlucky so set $d_0 = d_x$, $\widehat{H} = Hp$, $M = p$ and **goto** LOOP.

Chinese-Remaindering

9 Set $Hold = \widehat{H}$. Solve $\{\widehat{H} \equiv Hold \mod M$ and $\widehat{H} \equiv \widehat{H}p \mod p\}$ for $\widehat{H}$. Set $M = M \times p$.
If $\widehat{H} \neq Hold$ then **goto** LOOP.

Termination.

10 Set $\widetilde{H} = K_r^{-1} \widehat{H}(x, y)$ and let $\widetilde{H} = \sum_{i=0}^{d_0} \widetilde{c}_i x_0^i$ where $\widetilde{c}_i \in \mathbb{Z}[x_1, x_2, \ldots, x_n]$.
11 Set $\widehat{G} = \widetilde{H} / \gcd(\widetilde{c}_0, \widetilde{c}_1, \ldots, \widetilde{c}_{d_0})$ ($\widehat{G}$ is the primitive part of $\widetilde{H}$).
12 If $\widehat{G}|A$ and $\widehat{G}|B$ then **output** $\widehat{G}$.
13 **goto** LOOP.

Algorithm PGCD($K_r A$, $K_r B$, $K_r \Gamma$, S, τ, M)

Inputs $K_r A, K_r B \in \mathbb{Z}[x, y]$, $K_r \Gamma \in \mathbb{Z}[y]$, S a set of smooth primes, a term bound $\tau \geq \max \#h_i$ and M a positive integer.

Output With probability $\geq \frac{1}{2}$ a prime p and polynomial $Hp \in \mathbb{Z}_p[x, y]$ satisfying $Hp = K_r(H) \mod p$ and p does not divide M.

1 Pick a prime p at random from S that is not bad and does not divide M.
2 Pick a random shift s such that $0 < s < p$ and any generator α for $\mathbb{Z}_p^*$.

Compute-and-scale-images:

3 For j from 0 to $2\tau - 1$ do
4 Compute $a_j = K_r A(x, \alpha^{s+j}) \mod p$ and $b_j = K_r B(x, \alpha^{s+j}) \mod p$.
5 If $\deg_x a_j < \deg_x K_r A$ or $\deg_x b_j < \deg_x K_r B$ then **output** FAIL (α^{s+j} is a bad evaluation point.)

6 Compute $g_j = \gcd(a_j, b_j) \in \mathbb{Z}_p[x]$ using the Euclidean algorithm and set $g_j = K_r\Gamma(\alpha^{s+j}) \times g_j \bmod p$.
End for loop.

7 Set $d_0 = \deg g_0(x)$. If $\deg g_j(x) \neq d_0$ for any $1 \leq j \leq 2\tau - 1$ **output** FAIL (unlucky evaluations).

Interpolate-coefficients:

8 For $i = 0$ to d_0 do

9 Run the Berlekamp-Massey algorithm on the coefficients of x^i in the images $g_0, g_1, \ldots, g_{2\tau-1}$ to obtain $\lambda_i(z)$ and set $\tau_i = \deg \lambda_i(z)$.

10 Compute the roots m_j of each $\lambda_i(z)$ in $\mathbb{Z}_p$. If the number of distinct roots of $\lambda_i(z)$ is not equal τ_i then **output** FAIL (the feedback polynomial is wrong due to undetected unlucky evaluations.)

11 Set $e_k = \log_\alpha m_k$ for $1 \leq k \leq \tau_i$ and let $\sigma_i = \{y^{e_1}, y^{e_2}, \ldots y^{e_{\tau_i}}\}$.

12 Solve the τ_i by τ_i shifted transposed Vandermonde system

$$\left\{ \sum_{k=1}^{\tau_i} (\alpha^{s+j})^{e_k} u_k = \text{coefficient of } x^i \text{ in } g_j(x) \textbf{ for } 0 \leq j < \tau_i \right\}$$

modulo p for u and set $h_i(y) = \sum_{k=1}^{\tau_i} u_k y^{e_k}$. Note: $(\alpha^{s+j})^{e_k} = m_k^{s+j}$

End for loop.

13 Set $Hp := \sum_{i=0}^{d_0} h_i(y)x^i$ and **output** (p, Hp).

We remark that we do not check for termination after each prime because computing the primitive part of $\widetilde{H}$ or doing the trial divisions $\widehat{G}|A$ and $\widehat{G}|B$ in Step 12 could be more expensive than algorithm PGCD. Instead algorithm MGCD waits until the Chinese remaindering stabilizes in Step 9 before proceeding to test for termination.

Theorem 3.6. *Let $N = \log_{pmin} \|2H\|$. So N primes in S are sufficient to recover the integer coefficients of H using Chinese remaindering. Let X be the number of calls that Algorithm MGCD makes to Algorithm PGCD. Then $E[X] \leq 2(N + 1)$.*

Proof. Because the Kronecker substitution K_r is not bad, and the primes p used in PGCD are not bad and the evaluations points $\{\alpha^{s+j} : 0 \leq j \leq 2\tau - 1\}$ used in PGCD are not bad, in Step 6 of Algorithm PGCD, $\deg g_j(x) \geq \deg_{x_0} G$ by Lemma 1.8. Therefore $d_0 \geq \deg_{x_0} H = \deg_{x_0} G$ throughout Algorithm MGCD and $\deg_x \widehat{H} = \deg_{x_0} \widehat{G} \geq \deg_{x_0} G$. Since A and B are primitive in x_0, if $\widehat{G}|A$ and $\widehat{G}|B$ then it follows that $\widehat{G} = G$, so if algorithm MGCD terminates, it outputs G.

To prove termination observe that Algorithm MGCD proceeds in two phases. In the first phase MGCD loops while $d_0 > \deg_{x_0} H$. In this phase no useful work is accomplished. Observe that the loops in PGCD are of fixed length 2τ and d_0+1 so PGCD always terminates and algorithm MGCD tries another prime. Because at least 3/4 of the primes in S are good, and, for each prime, at least 3/4 of the possible evaluation point sequences are good, eventually algorithm PGCD will choose a good prime and a good evaluation point sequence after which $d_0 = \deg_{x_0} H$.

In the second phase MGCD loops using images Hp with $\deg_x Hp = d_0$ to construct $\widehat{H}$. Because the images $g_j(x)$ used satisfy $\deg_x g_j(x) = d_0 = \deg_{x_0} H$ and we scale them with $\Gamma(\alpha^{s+j})$, PGCD interpolates $Hp = H \mod p$ thus we have modular images of H. Eventually $\widehat{H} = H$ and the algorithm terminates.

Because the probability that the prime chosen from S is good is at least 3/4 and the evaluations points α^{s+j} are all good is at least 3/4, the probability that PGCD outputs a good image of H is at least 1/2. Since we need N images of H to recover H and one more to stabilize (see Step 9), $E[X] \leq 2(N+1)$ as claimed. □

4 FASTER ALGORITHM

In this section we consider the practical design of algorithms MGCD and PGCD. We make three improvements. Unfortunately, each improvement leads to a major complication.

4.1 Term Bounds

Recall that $H = \Delta G = \sum_{i=0}^{dG} h_i(x_1, \ldots, x_n) x_0^i$. Algorithms MGCD and PGCD assume a term bound τ on $\#h_i(y)$. In practice, good term bounds are usually not available. For the GCD problem, one cannot even assume that $\#G \leq \min(\#A, \#B)$ so we must modify the algorithm to compute $t_i = \#h_i(y)$.

We will follow [Kaltofen et al. 2000] which requires $2t_i + O(1)$ evaluation points to determine t_i with high probability. That is, we will loop calling the Berlekamp-Massey algorithm after $2, 4, 6, 8, \ldots,$ evaluation points and wait until we get two consecutive zero discrepancies, equivalently, we wait until the output $\lambda_i(z)$ does not change. This means $\lambda_i(z)$ is correct with high probability when p is sufficiently large. We give details in section 4.5. This loop will only terminate, however, if the sequence of points is generated by a polynomial and therein lies a problem.

Example 4.1. Consider the following GCD problem in $\mathbb{Z}[x, y]$. Let p be a prime and let

$$G = 1,\ \bar{A} = (yx+1)((y+1)x+2),\ \bar{B} = (yx+2)(y+p+1)x+2).$$

Observe that $LC(A) = y(y+1)$, $LC(B) = y(y+p+1)$, $\Gamma = y$ and $\gcd(A \mod p, B \mod p) = (y+1)x+2$ so p is unlucky.

Suppose we run algorithm PGCD with inputs $A = G\bar{A}$, $B = G\bar{B}$ and $\Gamma = y$ and suppose PGCD selects the prime p. Let $F(x, y) = x + \frac{2}{y+1}$. Algorithm PGCD will compute monic images $g_j(x) = F(x, \alpha^{s+j}) \mod p$ which after scaling by $\Gamma = \alpha^{s+j}$ are images of $yx + \frac{2y}{y+1}$ which is not a polynomial in y. So the Berlekamp-Massey algorithm will likely not stabilize and algorithm PGCD will loop trying to compute $\lambda_0(z)$. The problem is that scaling by $\Gamma = y$ does not result in a polynomial. We note that the same problem may be caused by an unlucky Kronecker substitution.

Our solution is to scale with either $\Gamma = LC(A)$ or $\Gamma = LC(B)$, whichever has fewer terms. Then, assuming p is not bad, $LC(\gcd(A \mod p, B \mod p))$ must divide both $LC(A) \mod p$ and $LC(B) \mod p$ thus scaling $g_j(x)$ by $LC(A)(\alpha^{s+j}) \mod p$ or $LC(B)(\alpha^{s+j}) \mod p$ will always give an image of a polynomial. The downside of this solution is that it may increase $t_i = \#h_i(y)$.

Another difficulty caused by $\lambda_i(z)$ stabilizing too early is that the support σ_i of $K_r(h_i)$ computed in Step 11 of PGCD may be wrong. We consider an example.

Example 4.2. Consider the following GCD problem in $\mathbb{Z}[x, y]$. Let p and q be prime and

$$G = x + py + qy^2 + py^4,\ \bar{A} = 1,\ \bar{B} = 1.$$

Suppose MGCD chooses p first and suppose PGCD returns $x + qy^2 \mod p$ so that $\sigma_0 = \{y^2\}$. Suppose MGCD chooses q next and suppose $\lambda_0(z)$ stabilizes too early and $\sigma_0 = \{y^3\}$ which is wrong. This could also be due to a missing term, for example, if $G = x + pqy + qy^2 + py^3$. If we combine these two images modulo p and q using Chinese remaindering to obtain $\widehat{H}$ of the form $x + \cdot y^2 + \cdot y^3$ we have a bad image in $\widehat{H}$ and we need somehow to detect it. Once detected, we do not want to restart the entire algorithm because we might be throwing away a lot of good images in $\widehat{H}$. Our solution in Steps 7–10 of algorithm MGCD1 is probabilistic.

4.2 Using smaller primes

Another consideration is the size of the primes that we use. We have implemented our GCD algorithm for 63 bit primes and 127 bit primes. By choosing a Kronecker substitution that is a priori good, and requiring that the 2τ evaluation points are good, the primes in S must be greater than $4\tau(2d + 2)(2d^2 + 1)^n$ where d bounds the degree of A and B in all variables. If instead we choose $r_i > \deg_{x_i} H$ then we will still be able to recover H from $K_r(H)$ but K_r may be unlucky.

Since $\deg_{x_i} H \le \min(\deg_{x_i} A, \deg_{x_i} B) \le d$, using $r_i = d + 1$ we replace the factor $(2d^2 + 1)^n$ with $(d + 1)^n$. We will detect if K_r is unlucky when $\deg g_j(x) > d_0$ by computing $d_0 = DegreeBound(A, B, 0)$ periodically (see Step 6 of MGCD1) so that eventually we obtain $d_0 = \deg_{x_0} G$ and can detect unlucky K_r. Once detected we will increase r_i by 1 to try a larger Kronecker substitution.

Recall that p is an unlucky prime if $p|R$ where $R = \mathrm{res}_{x_0}(\bar{A}, \bar{B})$. Because the inputs A and B are primitive in x_0 it follows that the integer coefficients of $\bar{A}$ and $\bar{B}$ are relatively prime. Therefore, the integer coefficients of R are also likely to have a very small common factor like 2. Thus the expected number of unlucky primes is very close to 0. In Theorem 2.13 we showed that the expected number of unlucky evaluations is 1 hence instead of using $p > 4\tau(2d + 2)(d + 1)^n$ we first try a prime $p > 4(d + 1)^n$. Should we encounter bad or unlucky evaluation points we will increase the length of p until we don't. This reduces the length of the primes for most inputs by at least a factor of 2.

Example 4.3. For our benchmark problem where $n = 8$, $d = 20$ and $\tau = 1000$ we have $\log_2[4\tau(2d + 2)(2d^2 + 1)^n] = 94.5$ bits which precludes our using 63 bit primes. On the other hand $\log_2[4(d + 1)^n] = 37.1$ bits, meaning a 63 bit prime is more than sufficient.

4.3 Using fewer evaluation points

Let $K_r(h_i) = \sum_{j=1}^{t_i} c_{ij} y^{e_{ij}}$ for some coefficients $c_{ij} \in \mathbb{Z}$ and exponents e_{ij} so that $\mathrm{Supp}(K_r(h_i)) = \{y^{e_{ij}} : 1 \le j \le t_i\}$. Because of the size of the primes chosen by algorithm MGCD, it is likely that the first good image Hp computed by PGCD has the entire support of $K_r(H)$, that is, $\mathrm{Supp}(\widehat{h_i}) = \mathrm{Supp}(K_r(h_i))$. Assuming this to be so, we can compute the next image of $K_r(H)$ modulo p using only t evaluations instead of $2t + O(1)$ as follows. We choose a prime p and compute $g_j(x)$ for $0 \le j < t$ as before in PGCD. Assuming these t images are all good, one may solve the the t_i by t_i shifted transposed Vandermonde systems

$$\left\{ \sum_{k=1}^{t_i} (\alpha^{s+j})^{e_{ij}} u_{ij} = \text{coefficient of } x^i \text{ in } g_j(x) \text{ for } 0 \le j \le \tau_i - 1 \right\}$$

for the unknown coefficients u_{ij} obtaining $Hp = \sum_{i=0}^{d_0} \sum_{j=0}^{t_i} u_{ij} y^{e_{ij}}$.

It is possible that the prime p used in PGCD may divide a coefficient c_{ij} in $K_r(H)$ in which case we will need to call PGCD again to compute more of the support of $K_r(H)$.

Definition 4.4. Let $f = \sum_{i=0}^{d} c_i y^{e_i}$ be a polynomial in $\mathbb{Z}[y]$. We say a prime p causes missing terms in f if p divides any coefficient c_i in f.

Our strategy to detect when $\text{Supp}(\widehat{h}_i) \not\subset \text{Supp}(K_r(h_i))$ is probabilistic. We compute one more image $j = \tau_i$ and check that the solutions of the Vandermonde systems are consistent with this image. Thus we require $t + 1$ evaluations instead of $2t + O(1)$. Once missing terms are detected, we call PGCD again to determine $\text{Supp}(K_r(h_i))$.

4.4 Algorithm MGCD1

We now present our algorithm as algorithm MGCD1 which calls subroutines PGCD1 and SGCD1. Like MGCD, MGCD1 loops calling PGCD1 to determine the $Hp = K_r(H) \mod p$. Instead of calling PGCD1 for each prime, MGCD1 after PGCD1 returns an image Hp, MGCD1 assumes the support of $K_r(H)$ is now known and uses SGCD1 for the remaining images.

Algorithm MGCD1(A, B)

Inputs $A, B \in \mathbb{Z}[x_0, x_1, \ldots, x_n]$ satisfying $n > 0$, A and B are primitive in x_0, and $\deg_{x_0} A > 0$, $\deg_{x_0} B > 0$.

Output $G = \gcd(A, B)$.

1 If $\#LC(A) < \#LC(B)$ set $\Gamma = LC(B)$ else set $\Gamma = LC(A)$.
2 Call Algorithm DegreeBound(A,B,i) to get $d_i \ge \deg_{x_i} G$ for $0 \le i \le n$.
If $d_0 = 0$ **return 1**.
3 Set $r_i = \min(\deg_{x_i} A, \deg_{x_i} B, d_i + \deg_{x_i}(\Gamma))$ for $1 \le i \le n$.
Set $\delta = 1$.

Kronecker-Prime

4 Set $r_i = r_i + 1$ for $1 \le i < n$. Let $Y = (y, y^{r_1}, y^{r_1 r_2}, \ldots, y^{r_1 r_2 \cdots r_{n-1}})$.
Set $K_r A = A(x, Y)$, $K_r B = B(x, Y)$ and $K_r \Gamma = \Gamma(Y)$.
If K_r is bad **goto** Kronecker-Prime otherwise set $\delta = \delta + 1$.

RESTART

5 Set $\widehat{H} = 0$, $M = 1$ and MissingTerms = true.
Set $\sigma_i = \phi$ and $\tau_i = 0$ for $0 \le i \le d_0$.

LOOP: // Invariant: $d_0 \ge \deg_{x_0} H$.

6 Compute $dx = DegreeBound(A, B, 0)$.
If $dx < d_0$ set $d_0 = dx$ and **goto** RESTART.
7 For each prime $p|M$ do // check current images
8 Set $a = K_r A \mod p$, $b = K_r B \mod p$ and $h = \widehat{H} \mod p$.
9 Pick β from $[0, p-1]$ at random.
10 If $K_r\Gamma(\beta) \neq 0$ and either $h(x,\beta)$ does not divide $a(x,\beta)$ or does not divide $b(x,\beta)$ then h is wrong so set $M = M/p$ and $\widehat{H} = \widehat{H} \mod M$ to remove this image.
End for loop.

If MissingTerms then // for first iteration
11 Pick a new smooth prime $p > 2^\delta \prod_{i=1}^n r_i$ that is not bad.
12 Call PGCD1(K_rA, K_rB, $K_r\Gamma$, d_0, τ, r, p).
13 If PGCD1 returned UNLUCKY($dmin$) set $d_0 = dmin$ and **goto** RESTART.
If PGCD1 returned FAIL **goto** Kronecker-Prime.
14 Let $\widehat{H}p = \sum_{i=0}^{d_0} \widehat{h}_i(y)x^i$ be the output of PGCD1.
Set MissingTerms = false, $\sigma_i := \sigma_i \cup \mathrm{Supp}(\widehat{h}_i)$ and $\tau_i = |\sigma_i|$ for $0 \le i \le d_0$.
else
15 Pick a new prime $p > 2^\delta \prod_{i=1}^n r_i$ that is not bad.
16 Call SGCD1(K_rA, K_rB, $K_r\Gamma$, d_0, σ, τ, p).
17 If SGCD1 returned UNLUCKY($dmin$) set $d_0 = dmin$ and **goto** RESTART.
If SGCD1 returned FAIL **goto** Kronecker-Prime.
If SGCD1 returned MISSINGTERMS set $\delta = \delta + 1$, Missingterms = true and **goto** LOOP.
18 Let $\widehat{H}p = \sum_{i=0}^{d_0} \widehat{h}_i(y)x^i$ be the output of SGCD1.
End If

Chinese-Remaindering
19 Set $Hold = \widehat{H}$. Solve $\{\widehat{H} \equiv Hold \mod M$ and $\widehat{H} \equiv \widehat{H}p \mod p\}$ for $\widehat{H}$. Set $M = M \times p$.
If $\widehat{H} \neq Hold$ then **goto** LOOP.

Termination.
20 Set $\widetilde{H} = K_r^{-1}\widehat{H}(x,y)$. Let $\widetilde{H} = \sum_{i=0}^{d_0} \widetilde{c}_i x_0^i$ where $\widetilde{c}_i \in \mathbb{Z}[x_1, \ldots, x_n]$.
21 Set $\widehat{G} = \widetilde{H}/\gcd(\widetilde{c}_0, \widetilde{c}_1, \ldots, \widetilde{c}_{d_0})$ ($\widehat{G}$ is the primitive part of $\widetilde{H}$).
22 If $\deg \widehat{G} \le \deg A$ and $\deg \widehat{G} \le \deg B$ and $\widehat{G}|A$ and $\widehat{G}|B$ then **return** $\widehat{G}$.
23 **goto** LOOP.

Algorithm PGCD1(K_rA, K_rB, $K_r\Gamma$, d_0, τ, r, p)
Inputs $K_rA, K_rB \in \mathbb{Z}[x,y]$ and $K_r\Gamma \in \mathbb{Z}[y]$, $d_0 \ge \deg_{x_0} G$ where $G = \gcd(A,B)$, term bound estimates $\tau \in \mathbb{Z}^{d_0+1}$, $r \in \mathbb{Z}^n$, and a smooth prime p.
Output $Hp \in \mathbb{Z}_p[x,y]$ satisfying $Hp = K_r(H) \bmod p$ or FAIL or UNLUCKY(dmin).
1 Pick a random shift $s \in \mathbb{Z}_p^*$ and any generator α for $\mathbb{Z}_p^*$.
2 Set $T = 0$.
LOOP

3 For j from $2T$ to $2T+1$ do

4 Compute $a_j = K_rA(x, \alpha^{s+j}) \bmod p$ and $b_j = K_rB(x, \alpha^{s+j}) \bmod p$.

5 If $\deg_x a_j < \deg_x K_rA$ or $\deg_x b_j < \deg_{x_0} K_rB$ then return FAIL (α^{s+j} is a bad evaluation point.)

6 Compute $g_j = \gcd(a_i, b_i) \in \mathbb{Z}_p[x]$ using the Euclidean algorithm.
Make g_j monic and set $g_j = K_r\Gamma(\alpha^{s+j}) \times g_j \bmod p$.

End for loop.

7 Set $dmin = \min \deg g_j(x)$ and $dmax = \max \deg g_j$ for $2T \le j \le 2T+1$.
If $dmin < d_0$ **output** UNLUCKY(dmin).
If $dmax > d_0$ **output** FAIL.

8 Set $T = T+1$.
If $T < \#K_r\Gamma$ or $T < \max_{i=0}^{d_0} \tau_i$ **goto** LOOP.

9 For i from 0 to d_0 do

10 Run the Berlekamp-Massey algorithm on the coefficients of x^i in the images $g_0, g_1, \ldots, g_{2T-1}$ to obtain $\lambda_i(z)$ and set $\tau_i = \deg \lambda_i(z)$. If either of the last two discrepancies were non-zero **goto** LOOP.

End for loop.

11 For i from 0 to d_0 do

12 Compute the roots m_k of $\lambda_i(z)$. If $\lambda_i(0) = 0$ or the number of distinct roots of $\lambda_i(z)$ is not equal τ_i then **goto** LOOP ($\lambda_i(z)$ stabilized too early)

13 Set $e_k = \log_\alpha m_k$ for $1 \le k \le \tau_i$ and let $\sigma_i = \{y^{e_1}, y^{e_2}, \ldots y^{e_{\tau_i}}\}$.
If $e_k \ge \prod_{i=1}^n r_i$ then $e_k > \deg_y K_r(H)$ so **output** FAIL (either the $\lambda_i(z)$ stabilized too early or K_r or p or all evaluations are unlucky).

14 Solve the τ_i by τ_i shifted transposed Vandermonde system

$$\left\{ \sum_{k=1}^{\tau_i} (\alpha^{s+j})^{e_k} u_k = \text{coefficient of } x^i \text{ in } g_j(x) \textbf{ for } 0 \le j < \tau_i \right\}$$

modulo p for u and set $\widehat{h}_i(y) = \sum_{k=1}^{\tau_i} u_k y^{e_k}$. Note: $(\alpha^{s+j})^{e_k} = m_k^{s+j}$.

End for loop.

15 Set $Hp = \sum_{i=0}^{d_0} \widehat{h}_i(y) x^i$ and **output** Hp.

The main **for** loop in Step 3 of algorithm PGCD1 evaluates K_rA and K_rB at α^{s+j} for $j = 2T$ and $j = 2T+1$ in Step 4 and computes their gcd in Step 6, that is, it computes two images before running the Berlekamp-Massey algorithm in Step 10. In our parallel implementation of algorithm PGCD1, for a multi-core computer with $N > 1$ cores, we compute N images at a time in parallel. We discuss this in Section 5.1.

Algorithm SGCD1(K_rA, K_rB, $K_r\Gamma$, d_0, σ, τ, p)

Inputs $K_rA, K_rB \in \mathbb{Z}[x, y]$, $K_r\Gamma \in \mathbb{Z}[y]$, $d_0 \ge \deg_{x_0} G$ where $G = \gcd(A, B)$, supports σ_i for $K_r(h_i)$ and $\tau_i = |\sigma_i|$, a smooth prime p.

Output FAIL or UNLUCKY(dmin) or MISSINGTERMS or $Hp \in \mathbb{Z}_p[x, y]$ satisfying if $d_0 = \deg_{x_0} G$ and $\sigma_i = \mathrm{Supp}(K_r(h_i))$ then $Hp = K_r(H) \mod p$.

1 Pick a random shift s such that $0 < s < p$ and any generator α for $\mathbb{Z}_p^*$.

2 Set $T = \max_{i=1}^{d_0} \tau_i$.
3 For j from 0 to T do // includes 1 check point
  4 Compute $a_j = K_r A(x, \alpha^{s+j}) \bmod p$ and $b_j = K_r B(x, \alpha^{s+j}) \bmod p$.
  5 If $\deg_x a_j < \deg_x K_r A$ or $\deg_x b_j < \deg_{x_0} K_r B$ then **output** FAIL (α^{s+j} is a bad evaluation point.)
  6 Compute $g_j = \gcd(a_i, b_i) \in \mathbb{Z}_p[x]$ using the Euclidean algorithm.
  Make g_j monic and set $g_j = K_r \Gamma(\alpha^{s+j}) \times g_j \bmod p$.
  End for loop.
6 Set $dmin = \min \deg g_j(x)$ and $dmax = \max \deg g_j$ for $0 \leq j \leq T$.
  If $dmin < d_0$ **output** UNLUCKY($dmin$).
  If $dmax > d_0$ **output** FAIL.
7 For i from 0 to d_0 do
  8 Let $\sigma_i = \{y^{e_1}, y^{e_2}, \ldots, y^{e_{\tau_i}}\}$.
  Solve the τ_i by τ_i shifted transposed Vandermonde system

$$\left\{ \sum_{k=1}^{\tau_i} u_k (\alpha^{s+j})^{e_k} = \text{coefficient of } x^i \text{ in } g_j(x) \textbf{ for } 0 \leq j \leq \tau_i - 1 \right\}$$

  modulo p for u and set $\widehat{h}_i(y) = \sum_{k=1}^{\tau_i} u_k y^{e_k}$. Note $(\alpha^{s+j})^{e_k} = m_k^{s+j}$.
  9 If $\widehat{h}_i((\alpha)^{s+\tau_i}) \neq$ coefficient of x^i in g_{τ_i} then **output** MISSINGTERMS.
  End for loop.
10 Set $Hp = \sum_{i=0}^{d_0} \widehat{h}_i(y) x^i$ and **output** Hp.

We prove that algorithm MGCD1 terminates and outputs $G = \gcd(A, B)$. We first observe that because MGCD1 avoids bad Kronecker substitutions and bad primes, and because the evaluation points α^{s+j} used in PGCD1 and SGCD1 are not bad, we have $K_r(\Gamma)(\alpha^{s+j}) \neq 0$ and $\deg g_j(x) \geq \deg_{x_0} G$ by Lemma 1.8. Hence $\deg_x \widehat{H} = \deg_{x_0} \widehat{G} \geq \deg_{x_0} G$. Therefore, if algorithm MGCD1 terminates, the conditions A and B are primitive and $\widehat{G}|A$ and $\widehat{G}|B$ imply $\widehat{G} = G$.

To prove termination we observe that Algorithm MGCD1 proceeds in four phases. In the first phase MGCD1 loops while $d_0 > \deg_x K_r(H) = \deg_{x_0} G$. Because Γ is either $LC(A)$ or $LC(B)$, even if K_r or p or all evaluations points are unlucky, the scaled images in Step 6 of algorithm PGCD1 are images of a polynomial in $\mathbb{Z}[x, y]$ hence the $\lambda_i(z)$ polynomials must stabilize and algorithm PGCD1 always terminates.

Now if PGCD1 or SGCD1 output UNLUCKY($dmin$) then d_0 is decreased, otherwise, they output FAIL or MISSINGTERMS or an image Hp and MGCD1 executes Step 6 at the beginning of the main loop. Eventually the call to DegreeBound in Step 6 will set $d_0 = \deg_{x_0} G$ after which unlucky Kronecker substitutions, unlucky primes and unlucky evaluation points can be detected.

Suppose $d_0 = \deg_{x_0} G$ for the first time. In the second phase MGCD1 loops while PGCD1 outputs FAIL due to an unlucky Kronecker substitution or an unlucky prime or bad or unlucky evaluation points or the Berlekamp-Massey algorithm stabilized too early. If PGCD1 outputs FAIL, since we don't know if this is due to an unlucky Kronecker substitution or an unlucky prime p, MGCD1 increases r_i by 1 and the size of p by 1 bit. Since there are only finitely many unlucky K_r, eventually K_r will be lucky. And since there are only finitely many unlucky primes, eventually p be lucky. Finally, since we keep increasing the length of p, eventually p will be sufficiently large so that no bad or unlucky evaluations are encountered in PGCD1 and the Berlekamp-Massey algorithm does not stabilize too early. Then PGCD1 succeeds and outputs an image Hp with $\deg_x Hp = d_0 = \deg_{x_0} G$.

In the third phase MGCD1 loops while the $\sigma_i \not\supseteq K_r(h_i)$, that is, we don't yet have the support for all $K_r(h_i) \in \mathbb{Z}[y]$ either because of missing terms or because a $\lambda_i(z)$ polynomial stabilized too early in PGCD1, and went undetected.

We now prove that Step 9 of SGCD1 detects that $\sigma_i \not\supseteq \mathrm{Supp}(K_r(h_i))$ with probability at least $\frac{3}{4}$ so that PGCD1 is called again in MGCD1.

Suppose $\sigma_i \not\supseteq \mathrm{Supp}(K_r(h_i))$ for some i. Consider the first τ_i equations in Step 8 of SGCD1. We first argue that this linear system has a unique solution. Let $m_k = \alpha^{e_k}$ so that $(\alpha^{s+j})^{e_k} = m_k^{s+j}$. The coefficient matrix W of the linear system has entries

$$W_{jk} = m_k^{s+j-1} \text{ for } 1 \le j \le \tau_i \text{ and } 1 \le k \le \tau_i.$$

W is a shifted transposed Vandermonde matrix with determinant

$$\det W = m_1^s \times m_2^s \times \cdots \times m_{\tau_i}^s \times \prod_{1 \le j < k \le \tau_i} (m_j - m_k).$$

Since $m_k = \alpha^{e_k}$ we have $m_k \neq 0$ and since $p > \deg_y K_r(H)$ the m_k are distinct hence $\det W \neq 0$ and the linear system has a unique solution for u.

Let $E(y) = \phi_p(K_r(h_i)(y)) - \widehat{h}_i(y)$ where $\widehat{h}_i(y) = \sum_{k=1}^{\tau_i} u_k y^{e_k}$ is the polynomial in $\mathbb{Z}_p[y]$ computed in Step 8 of SGCD1. It satisfies $E(\alpha^{s+j}) = 0$ for $0 \le j < \tau_i$. If $\sigma_i \not\supseteq \mathrm{Supp}(K_r(h_i))$ then $E(y) \neq 0$ and algorithm SGCD1 tests for this in Step 9 when it checks if $E(\alpha^{s+\tau_i}) \neq 0$. It is possible, however, that $E(\alpha^{s+\tau_i}) = 0$. We bound the probability that this can happen.

Lemma 4.5. *If s is chosen at random from $[1, p-1]$ then*

$$\mathrm{Prob}[E(\alpha^{s+\tau_i}) = 0] \; < \; \frac{1}{4}.$$

Proof. The condition in Step 13 of algorithm PGCD1 means $\deg \bar{h}_i(y) < \prod_{j=1}^n r_j$ hence $\deg_y(E) < \prod_{j=1}^n r_j$. Now s is chosen at random so $\alpha^{s+\tau_i}$ is random on $[1, p-1]$ therefore

$$\mathrm{Prob}[E(\alpha^{s+\tau_i}) = 0] \le \frac{\deg_y(E)}{p-1} < \frac{\prod_{j=1}^n r_j}{p-1}.$$

Since the primes in SGCD1 satisfy $p > 4\prod_{j=1}^n r_j$ the result follows. □

Thus eventually $\sigma_i \not\supseteq \mathrm{Supp}(K_r(h_i))$ is detected in Step 9 of algorithm SGCD1. Because we cannot tell whether this is caused by missing terms or $\lambda_i(z)$ stabilizing too early and going undetected in Steps 12 and 13 of PGCD1, we increase the size of p by 1 bit in Step 17 so that with repeated calls to PGCD1, $\lambda_i(z)$ will eventually not stabilize early and we obtain $\sigma_i \supseteq \mathrm{Supp}(K_r(h_i)) \mod p$.

How many good images are needed before $\sigma_i \supseteq \mathrm{Supp}(K_r(h_i))$ for all $0 \le i \le d_0$? Let $pmin$ be the smallest prime used by algorithm PGCD1. Let $N = \lfloor \log_{pmin} \|K_r(H)\| \rfloor$. Since at most N primes $\geq pmin$ can divide any integer coefficient in $K_r(H)$ then $N+1$ good images from PGCD1 are sufficient to recover the support of $K_r(H)$.

In the fourth and final phase MGCD1 loops calling SGCD1 while $\widehat{H} \neq K_r(H)$. If SGCD1 outputs an image Hp then since $d_0 = \deg_{x_0} H$ and $\sigma_i \supseteq \text{Supp}(K_r(h_i))$ then Hp satisfies $Hp = H \mod p$. The image is combined with previously computed images in $\widehat{H}$ using Chinese remaindering. But as noted in example 4.2, $\widehat{H}$ may contain a bad image. A bad image arises because either PGCD1 returns a bad image Hp because a $\lambda_i(z)$ stabilized too early or because SGCD1 uses a support with missing terms and fails to detect it.

Consider the prime p and polynomial $h(x,y)$ in Step 8 of MGCD1. Suppose $h(x,y)$ is a bad image, that is, $h \neq K_r(H) \mod p$. We claim Steps $7-10$ of MGCD1 detect this bad image with probability at least $1/2$ and since the test for a bad image is executed repeatedly in the main loop, algorithm MGCD1 eventually detects it and removes it hence eventually MGCD1 computes $K_r(H)$ and terminates with output G.

To prove the claim recall that $H = \Delta G$ and $LC(H) = \Gamma$. Because Step 8 of PGCD1 requires $T \geq \#K_r(\Gamma)$ this ensures algorithm PGCD1 always outputs Hp with $LC(Hp) = K_r(\Gamma) \mod p$ hence $LC(h) = K_r(\Gamma) \mod p$.

If $h = K_r(H) \mod p$ and $K_r(\Gamma)(\beta) \neq 0$ then in Step 10 of MGCD1 $h(x,\beta)$ must divide $a(x,\beta)$ and divide $b(x,\beta)$ as $a(x,\beta) = K_r(A)(x,\beta)$ and $b(x,\beta) = K_r(B)(x,\beta)$. Now suppose $h \neq K_r(H) \mod p$. Then Step 10 of MGCD1 fails to detect this bad image if $K_r\Gamma(\beta) \neq 0$ and $h(x,\beta)|a(x,\beta)$ and $h(x,\beta)|b(x,\beta)$ in $\mathbb{Z}_p[x]$. Since $\deg_x h = d_0 = \deg_x K_r(H)$ it must be that $h(x,\beta)$ is an associate of $K_r(H)(x,\beta)$. But since $LC(h) = K_r(\Gamma) \mod p = LC(K_r(H)) \mod p$ we have $h(x,\beta) = K_r(H)(x,\beta) \mod p$. Let $E = h - K_r(H) \mod p$. Therefore the test for a bad image h succeeds iff $K_r(\Gamma)(\beta) \neq 0$ and $E(x,\beta) \neq 0$. Lemma 4.6 below implies the test succeeds with probability at least $1/2$.

Lemma 4.6. *If β is chosen at random from $[0, p-1]$ then*

$$Prob[K_r(\Gamma)(\beta) \neq 0] \geq \frac{3}{4} \text{ and } Prob[E(x,\beta) \neq 0] \geq \frac{3}{4}.$$

Proof. The primes p chosen in Step 15 of MGCD1 satisfy $p > 2^\delta \prod_{i=1}^n r_i$ with $\delta \geq 2$. Since $\deg_y K_r(\Gamma) < \prod_{i=1}^n r_i$ by Step 3 of MGCD1 then $Prob[K_r(\Gamma)(\beta) \mod p = 0] \leq \frac{\deg_y(\Gamma)}{p} < \frac{1}{4}$. Since $\deg_y h < \prod_{i=1}^n r_i$ by Step 13 of PGCD 1 and since r_i is chosen in Step 3 of MGCD1 so that $r_i \geq \deg_{x_i} H$ we have $\deg_y K_r(H) < \prod_{i=1}^n r_i$. Hence $\text{Prob}[E(x,\beta) = 0] \leq \frac{\deg_y E}{p} < \frac{1}{4}$. □

4.5 Determining t

Algorithm PGCD1 tests in Steps 9 and 10 if both of the last two discrepancies are 0 before it executes Step 11. But it is possible that in Step 11 $\tau_i < \#h_i$.

Let $V_r = (v_0, v_1, \ldots, v_{2r-1})$ be a sequence where $r \geq 1$. The Berlekamp-Massey algorithm (BMA) with input V_r computes a feedback polynomial $c(z)$ which is the reciprocal of $\lambda(z)$ if $r = t$. In PGCD1, we determine the t by computing $c(z)$s on the input sequence V_r for $r = 1, 2, 3, \ldots$. If a $c(z)$ remains unchanged from the input V_k to the input V_{k+1}, then we conclude that this $c(z)$ is *stable* which implies that the last two consecutive discrepancies are both zero, see [Kaltofen et al. 2000; Massey 1969] for a definition of the discrepancy. However, it is possible that the degree of $c(z)$ on the input V_{k+2} might increase again. In [Kaltofen et al. 2000], Kaltofen, Lee and Lobo proved (Theorem 3) that the BMA encounters the first zero discrepancy after $2t$ points with probability at least

$$1 - \frac{t(t+1)(2t+1)\deg(C)}{6|S|}$$

where S is the set of all possible evaluation points. Here is an example where we encounter a zero discrepancy before $2t$ points. Consider

$$f(y) = y^7 + 60y^6 + 40y^5 + 48y^4 + 23y^3 + 45y^2 + 75y + 55$$

over $\mathbb{Z}_{101}$ with generator $\alpha = 93$. Since f has 8 terms, 16 points are required to determine the correct $\lambda(z)$ and two more for confirmation. We compute $f(\alpha^j)$ for $0 \le j \le 17$ and obtain $V_9 = (44, 95, 5, 51, 2, 72, 47, 44, 21, 59, 53, 29, 71, 39, 2, 27, 100, 20)$. We run the BMA on input V_r for $1 \le r \le 9$ and obtain feedback polynomials in the following table.

r	Output $c(z)$
1	$69z + 1$
2	$24z^2 + 59z + 1$
3	$24z^2 + 59z + 1$
4	$24z^2 + 59z + 1$
5	$70z^7 + 42z^6 + 6z^3 + 64z^2 + 34z + 1$
6	$70z^7 + 42z^6 + 25z^5 + 87z^4 + 16z^3 + 20z^2 + 34z + 1$
7	$z^7 + 67z^6 + 95z^5 + 2z^4 + 16z^3 + 20z^2 + 34z + 1$
8	$31z^8 + 61z^7 + 91z^6 + 84z^5 + 15z^4 + 7z^3 + 35z^2 + 79z + 1$
9	$31z^8 + 61z^7 + 91z^6 + 84z^5 + 15z^4 + 7z^3 + 35z^2 + 79z + 1$

The ninth call of the BMA confirms that the feedback polynomial returned by the eighth call is the desired one. But, by our design, the algorithm terminates at the third call because the feedback polynomial remains unchanged from the second call. It also remains unchanged for V_4. In this case, $\lambda(z) = z^2 c(1/z) = z^2 + 59z + 24$ has roots 56 and 87 which correspond to monomials y^4 and y^{20} since $\alpha^4 = 56$ and $\alpha^{20} = 87$. The example shows that we may encounter a stable feedback polynomial too early.

5 IMPLEMENTATION AND OPTIMIZATIONS

5.1 Evaluation

Let $A, B \in \mathbb{Z}_p[x_0, x_1, \ldots, x_n]$, $s = \#A + \#B$, and $d = \max_{i=1}^n d_i$ where $d_i = \max(\deg_{x_i} A, \deg_{x_i} B)$. If we use a Kronecker substitution

$$K(A) = A(x, y, y^{r_1}, \ldots, y^{r_1 r_2 \cdots r_{n-1}}) \text{ with } r_i = d_i + 1,$$

then $\deg_y K(A) < (d+1)^n$. Thus we can evaluate the s monomials in $K(A)(x,y)$ and $K(B)(x,y)$ at $y = \alpha^k$ in $O(sn \log d)$ multiplications. Instead we first compute $\beta_1 = \alpha^k$ and $\beta_{i+1} = \beta_i^{r_i}$ for $i = 1, 3, \ldots, n-2$ then precompute n tables of powers $1, \beta_i, \beta_i^2, \ldots, \beta_i^{d_i}$ for $1 \le i \le n$ using at most nd multiplications. Now, for each term in A and B of the form $cx_0^{e_0} x_1^{e_1} \ldots x_n^{e_n}$ we compute $c \times \beta_1^{e_1} \times \cdots \times \beta_n^{e_n}$ using the tables in n multiplications. Hence we can evaluate $K(A)(x,\alpha^k)$ and $K(B)(x,\alpha^k)$ in at most $nd + ns$ multiplications. Thus for T evaluation points $\alpha, \alpha^2, \ldots, \alpha^T$, the evaluation cost is $O(ndT + nsT)$ multiplications.

When we first implemented algorithm PGCD we noticed that often well over 95% of the time was spent evaluating the input polynomials A and B at the points α^k. This happens when $\#H \ll \#A + \#B$. The following method uses the fact that for a monomial $M_i(x_1, x_2, \ldots, x_n)$

$$M_i(\beta_1^k, \beta_2^k, \ldots, \beta_n^k) \;=\; M_i(\beta_1, \beta_2, \ldots, \beta_n)^k$$

to reduce the total evaluation cost from $O(ndT + nsT)$ multiplications to $O(nd + ns + sT)$. Note, no sorting on x_0 is needed in Step 4b if the monomials in the input A are are sorted on x_0.

Algorithm Evaluate.

Input $A = \sum_{i=1}^m c_i x_0^{e_i} M_i(x_1, \ldots, x_n) \in \mathbb{Z}_p[x_0, \ldots, x_n]$, $T > 0$, $\beta_1, \beta_2, \ldots, \beta_n \in \mathbb{Z}_p$, and integers $d_1, d_2, \ldots, d_n$ with $d_i \geq \deg_{x_i} A$.

Output $y_k = A(x_0, \beta_1^k, \ldots, \beta_n^k)$ for $1 \leq k \leq T$.

1 Create the vector $C = [c_1, c_2, \ldots, c_m] \in \mathbb{Z}_p^m$.
2 Compute $[\beta_i^j : j = 0, 1, \ldots, d_i]$ for $1 \leq i \leq n$.
3 Compute $\Gamma = [M_i(\beta_1, \beta_2, \ldots, \beta_n) : 1 \leq i \leq m]$.
4 For $k = 1, 2, \ldots, T$ do
  4a Compute the vector $C := [C_i \times \Gamma_i$ for $1 \leq i \leq m]$.
  4b Assemble $y_k = \sum_{i=1}^m C_i x_0^{e_i} = A(x_0, \beta_1^k, \ldots, \beta_n^k)$.

The algorithm computes y_k as the matrix vector product.

$$\begin{bmatrix} \Gamma_1 & \Gamma_2 & \ldots & \Gamma_m \\ {\Gamma_1}^2 & {\Gamma_2}^2 & \ldots & {\Gamma_m}^2 \\ \vdots & \vdots & \vdots & \vdots \\ {\Gamma_1}^T & {\Gamma_2}^T & \ldots & {\Gamma_m}^T \end{bmatrix} \begin{bmatrix} c_1\, x_0^{e_1} \\ c_2\, x_0^{e_2} \\ c_3\, x_0^{e_3} \\ \vdots \\ c_m\, x_0^{e_m} \end{bmatrix} = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_T \end{bmatrix}.$$

Even with this improvement evaluation still takes most of the time so we must parallelize it. Each evaluation of A could be parallelized in blocks of size m/N for N cores. In Cilk C, this is only effective, however, if the blocks are large enough (at least 50,000) so that the time for each block is much larger than the time it takes Cilk to create a task. For this reason, it is necessary to also parallelize on k. To parallelize on k for N cores, we multiply the previous N values of C in parallel by the vector

$$\Gamma_N = [M_i(\beta_1, \beta_2, \ldots, \beta_n)^N : 1 \leq i \leq m]$$

Because most of the time is still in evaluation, we have considered the asymptotically fast method of van der Hoven and Lecerf [van der Hoven and Lecerf 2013] and how to parallelize it. For our evaluation problem it has complexity $O(nd + ns + s\log^2 T)$ which is better than our $O(nd+ns+sT)$ method for large T. In [Monagan and Wong 2017], Monagan and Wong implemented this method using 64 bit machine integers and in comparing it with our method used here, found the break even point to be around $T = 500$.

5.2 The non-monic case and homogenization.

Algorithm PGCD interpolates $H = \Delta G$ from scaled monic images $K(\Gamma)(\alpha^j)g_j(x)$ which are computed in Step 6. If the number of terms of Δ is m and $m > 1$ then it is likely that $\#H$ is greater than $\#G$, which means we need more evaluation points for sparse interpolation. For sparse inputs, this may increase t by a factor of m.

One such example occurs in multivariate polynomial factorization. Given a polynomial f in $\mathbb{Z}[x_0, x_1, \ldots, x_n]$, factorization algorithms first identify and remove repeated factors by doing a square-free factorization. See Section 8.1 of [Geddes et al. 1992]. The first Step of square-free factorization computes

$$g = \gcd(f, h = \frac{\partial f}{\partial x_0}).$$

Then we have $\Gamma = \gcd(LC(f), LC(h)) = \gcd(LC(f), dLC(f)) = LC(f)$ and $\Delta = LC(f)/LC(g)$ which can be a large polynomial.

Obviously, if either A or B is monic in x_i for some $i > 0$ then we may simply use x_i as the main variable our GCD algorithm instead of x_0 so that $\#\Gamma = \#\Delta = 1$. Similarly, if either A or B have a constant term in any x_i, that is, $A = \sum_{j=0} a_j x_i^j$ and $B = \sum_{j=0} b_j x_i^j$ and either a_0 or b_0 are integers, then we can reverse the coefficients of both A and B in x_i so that again $\#\Gamma = \#\Delta = 1$. But many multivariate GCD problems in practice do not satisfy any of these conditions.

Suppose A or B has a constant term. We propose to exploit this by homogenizing A and B. Let f be a non-zero polynomial in $\mathbb{Z}[x_1, x_2, \ldots, x_n]$ and

$$H_z(f) = f(\frac{x_1}{z}, \frac{x_2}{z}, \ldots, \frac{x_n}{z}) z^{\deg f}$$

denote the homogenization of f in z. We have the following properties of $H_z(f)$.

Lemma 5.1. *Let a and b be in $\mathbb{Z}[x_1, x_2, ..., x_n]$. For non-zero a and b*

(i) $H_z(a)$ is homogeneous in $z, x_1, \ldots, x_n$ of degree $\deg a$*,*
(ii) $H_z(a)$ is invertible: if $f(z) = H_z(a)$ then $H_z^{-1}(f) = f(1) = a$,
(iii) $H_z(ab) = H_z(a)H_z(b)$, and
(iv) $H_z(\gcd(a, b)) = \gcd(H_z(a), H_z(b))$.

Proof: To prove (i) let $M = x_1^{d_1} x_2^{d_2} \ldots x_n^{d_n}$ be a monomial in a and let $d = \deg a$. Then

$$H_z(M) = z^d \frac{x_1^{d_1}}{z^{d_1}} \ldots \frac{x_n^{d_n}}{z^{d_n}}.$$

Observe that since $d \geq d_1 + d_2 + \cdots + d_n$ then $\deg_z(H_z(M)) \geq 0$ and $\deg H_z(M) = d$. Properties (ii) and (iii) follow easily from the definition of H_z. To prove (iv) let $g = \gcd(a, b)$. Then $a = g\bar{a}$ and $b = g\bar{b}$ for some $\bar{a}, \bar{b}$ with $\gcd(\bar{a}, \bar{b}) = 1$. Now

$$\begin{aligned} \gcd(H_z(a), H_z(b)) &= \gcd(H_z(g\bar{a}),\ H_z(g\bar{b})) \\ &= \gcd(H_z(g)H_z(\bar{a}),\ H_z(g)H_z(\bar{b})) \text{ by (iii)} \\ &= H_z(g) \times \gcd(H_z(\bar{a}),\ H_z(\bar{b})) \text{ up to units.} \end{aligned}$$

Let $c(z) = \gcd(H_z(\bar{a}), H_z(\bar{b}))$ in $\mathbb{Z}[z, x_1, \ldots, x_n]$. It suffices to prove that $\gcd(\bar{a}, \bar{b}) = 1$ implies $c(z)$ is a unit. Now $c(z) = \gcd(H_z(\bar{a}), H_z(\bar{b})) \Rightarrow c(z)|H_z(\bar{a})$ and $c(z)|H_z(\bar{b})$ which implies

$$H_z(\bar{a}) = c(z)q(z) \text{ and } H_z(\bar{b}) = c(z)r(z)$$

for some $q, r \in \mathbb{Z}[z, x_1, \ldots, x_n]$. Applying H^{-1} to these relations we get $\bar{a} = c(1)q(1)$ and $\bar{b} = c(1)r(1)$. Now $\gcd(\bar{a}, \bar{b}) = 1$ implies $c(1)$ is a unit and thus $q(1) = \pm\bar{a}$ and $r(1) = \pm\bar{b}$. We need to show that $c(z)$ is a unit. Let $d = \deg H_z(\bar{a})$. Since $\deg H_z(\bar{a}) = \deg \bar{a}$ by (i) and $q(1) = \pm\bar{a}$ then $\deg q(1) = d$ and hence $\deg q(z) \geq d$. Now since $H_z(\bar{a}) = c(z)q(z)$ it must be that $\deg c(z) = 0$ and $\deg q(z) = d$. Since $c(1) = \pm 1$ then $\deg c(z) = 0$ implies $c(z) = \pm 1$. $\square$.

Properties (iii) and (iv) mean we can compute $G = \gcd(A, B)$ using

$$G = H_z^{-1} \gcd(H_z(A), H_z(B)).$$

Notice also that homogenization preserves sparsity. To see why homogenization may help we consider an example.

Example 5.2. Let $G = x^2 + y + 1$, $\bar{A} = xy + x + y + 1 = (y+1)x + (y+1) = (x+1)y + (x+1)$ and $\bar{B} = x^2y + xy^2 + x^2 + y^2 = (y+1)x^2 + y^2(x+1)$. Then $H_z(G) = z^2 + yz + x^2$, $H_z(\bar{A}) = z^2 + (x+y)z + xy$, and $H_z(\bar{B}) = (x^2 + y^2)z + (x^2y + xy^2)$.

Notice in Example 11 that A and B are neither monic in x nor monic in y but since A has a constant term, $H_z(A)$ is monic in z. If we use x as x_0 in Algorithm PGCD then $\Gamma = \gcd(y+1, y+1) = y+1 = \Delta$ and we interpolate $H = \Delta G = (y+1)x^2 + (y^2 + 2y + 1)$ and $t = 3$. If we use y as x_0 in Algorithm PGCD then $\Gamma = \gcd(x+1, x+1) = x+1 = \Delta$ and we interpolate $H = \Delta G = (x+1)y + (x^3 + x^2 + x + 1)$ and $t = 4$. But if we use use z as x_0 in Algorithm PGCD then $\Gamma = \gcd(1, x^2 + y^2) = 1$ hence $\Delta = 1$ and we interpolate $H_z(G) = z^2 + yz + x^2$ and $t = 1$.

If A or B has a constant term then because homogenizing A and B means $\Gamma \in \mathbb{Z}$ and $\Delta \in \mathbb{Z}$, we always homogenize if $\#\Gamma > 1$. There is, however, a cost to in homogenizing for the GCD problem, namely, we increase the number of variables to interpolate by 1 and we increase the cost of the univariate images in $\mathbb{Z}_p[z]$ if the degree increases. The degree may increase by up to a factor of $n+1$. For example, if $G = 1 + \prod_{i=0}^n x_i^{d-1}$, $\bar{A} = 1 + \prod_{i=0}^n x_i$ and $\bar{B} = 1 - \prod_{i=0}^n x_i$ then $\deg_{x_i} A = d = \deg_{x_i} B$ but $\deg_z H_z(A) = (n+1)d = \deg_z H_z(B)$. Homogenizing can also increase t when G has many terms of the same total degree.

5.3 Bivariate images

Recall that we interpolate $H = \sum_{i=0}^{dG} h_i(x_1, \ldots, x_n)x_0^i$ where $H = \Delta G$. The number of evaluation points used by algorithm PGCD is $2t + O(1)$ where $t = \max_{i=0}^{dG} \#h_i$. Since the cost of our algorithm is multiplied by the number of evaluation points needed we can reduce the cost of algorithm PGCD if we can reduce t.

Algorithm PGCD interpolates H from univariate images in $\mathbb{Z}_p[x_0]$. If instead we interpolate H from bivariate images in $\mathbb{Z}_p[x_0, x_1]$, this will likely reduce t when $\#\Delta = 1$ and when $\#\Delta > 1$. For our benchmark problem, where $\Delta = 1$, doing this reduces t from 1198 to 130 saving a factor of 9.2. On the other hand, we must now compute bivariate GCDs in

$\mathbb{Z}_p[x_0, x_1]$. To decide whether this will lead to an overall gain, we need to know and the cost of computing bivariate images and the likely reduction in t.

To compute a bivariate GCD in $\mathbb{Z}_p[x_0, x_1]$ we have implemented Brown's dense modular GCD algorithm from [Brown 1971]. If G is sparse, then for sufficiently large t and n, G is likely dense in x_0 and x_1, so using a dense GCD algorithm is efficient. The complexity of Brown's algorithm is $O(d^3)$ arithmetic operations in $\mathbb{Z}_p$ where $d = \max_{i=0}^{1}(\deg_{x_i} A, \deg_{x_i} B)$. Thus if this cost is less than the cost of evaluating the inputs, which using our evaluation algorithm from 3.2 is s multiplications in $\mathbb{Z}_p$ where $s = \#A + \#B$, then the cost of the bivariate images does not increase the overall cost of the algorithm significantly. For our benchmark problem, $s = 2 \times 10^6$ and $d^3 = 40^3 = 64,000$ so the cost of a bivariate image is negligible compared with the cost of an evaluation.

Let us write

$$H = \sum_{i=0}^{d_0} h_i(x_1, \ldots, x_n) x_0^i = \sum_{i=0}^{d_0} \sum_{j=0}^{d_1} h_{ij}(x_2, \ldots x_n) x_0^i x_1^j$$

and define $t_1 = \max \#h_i$ and $t_2 = \max \#h_{ij}$. The ratio t_1/t_2 is reduction of the number of evaluation points needed by our algorithm. The maximum reduction in t occurs when the terms in H are distributed evenly over the coefficients of H in x_1, that is, then $t_1/t_2 = 1 + d_1 = 1 + \deg_{x_1} \Delta + \deg_{x_1} G$. For some very sparse inputs, there is no gain. For example, for

$$H = x_0^d + x_1^d + x_2^d + \cdots + x_n^d + 1$$

we have $t_1 = n$ and $t_2 = n - 1$ and the gain is negligible.

If H has total degree D and H is dense then the number of terms in $h_i(x_1, \ldots, x_n)$ is $\binom{D-i+n}{n}$ which is a maximum for h_0 where $\#h_0 = \binom{D+n}{n}$. A conservative assumption is that $\#h_i$ is proportional to $\binom{n+D-i}{n}$ and similarly $\#h_{ij}$ is proportional to $\binom{n-1+D-(i+j)}{n-1}$. In this case, the reduction is a factor of

$$\frac{\#h_0}{\#h_{00}} = \binom{n+D}{n} \Big/ \binom{n-1+D}{n-1} = \frac{n+D}{n}.$$

For our benchmark problem where $n = 8$ and $D = 60$ this is $8.5 = \frac{68}{8}$.

6 BENCHMARKS

We have implemented algorithm PGCD for 31, 63 and 127 bit primes in Cilk C. For 127 bit primes we use the 128 bit signed integer type `__int128_t` supported by the gcc compiler. We parallelized evaluation (see Section 3.2) and we interpolate the coefficients $h_i(y)$ in parallel in Step 11 of Algorithm PGCD1.

The new algorithm requires $2t + \delta$ images (evaluation points) for the first prime and $t + 1$ images for the remaining primes. The additional image ($t + 1$ images instead of t) is used to check if the support of H (see Step 9 of Algorithm SGCD1) obtained from the first prime is correct.

To assess how good our new algorithm is, we have compared it with the serial implementations of Zippel's algorithm in Maple 2016 and Magma V2.22. For Maple we are able to determine the time spent computing G modulo the first prime in Zippel's algorithm. It is typically over 99% of the total GCD time. The reason for this is that Zippel's algorithm requires $O(ndt)$ images for the first prime but only $t + 1$ images for the remaining primes.

We also timed Maple's implementation of Wang's EEZ-GCD algorithm from [Wang 1980, 1978]. It was much slower than Zippel's algorithm on these inputs so we have not included timings for it. Note, older versions of Maple and Magma both used the EEZ-GCD algorithm for multivariate polynomial GCD computation.

All timings were made on the gaby server in the CECM at Simon Fraser University. This machine has two Intel Xeon E-2660 8 core CPUs running at 3.0 GHz on one core and 2.2 GHz on 8 cores. Thus maximum parallel speedup is a factor of $16 \times 2.2/3.0 = 11.7$.

6.1 Benchmark 1

For our first benchmark (see Table 3) we created polynomials $G, \bar{A}$ and $\bar{B}$ in 6 variables ($n = 5$) and 9 variables ($n = 8$) of degree at most d in each variable. We generated $100d$ terms for G and 100 terms for $\bar{A}$ and $\bar{B}$. That is, we hold t approximately fixed to test the dependence of the algorithms on d.

The integer coefficients of $G, \bar{A}, \bar{B}$ were generated at random from $[0, 2^{31} - 1]$. The monomials in $G, \bar{A}$ and $\bar{B}$ were generated using random exponents from $[0, d - 1]$ for each variable. For G we included monomials $1, x_0^d, x_1^d, \ldots, x_n^d$ so that G is monic in all variables and $\Gamma = 1$. Maple and Magma code for generating the input polynomials is given in the Appendix.

Our new algorithm used the 62 bit prime $p = 29 \times 2^{57} + 1$. Maple used the 32 bit prime $2^{32} - 5$ for the first image in Zippel's algorithm.

			New GCD algorithm		Zippel's algorithm	
n	d	t	1 core (eval)	16 cores	Maple	Magma
5	5	110	0.29s (64%)	0.074s (3.9x)	3.57s	0.60s
5	10	114	0.62s (68%)	0.091s (6.8x)	48.04s	6.92s
5	20	122	1.32s (69%)	0.155s (8.5x)	185.70s	296.06s
5	50	121	3.48s (69%)	0.326s (10.7x)	1525.80s	$> 10^5s$
5	100	123	7.08s (69%)	0.657s (10.8x)	6018.23s	NA
5	200	125	14.64s (71%)	1.287s (11.4x)	NA	NA
5	500	135	38.79s (71%)	3.397s (11.4x)	NA	NA
8	5	89	0.27s (61%)	0.065s (4.2x)	32.47s	2.28s
8	10	110	0.63s (65%)	0.098s (6.4x)	138.41s	7.33s
8	20	114	1.35s (66%)	0.163s (8.3x)	664.33s	78.77s
8	50	113	3.52s (66%)	0.336s (10.5x)	6390.22s	800.15s
8	100	121	7.43s (68%)	0.645s (11.5x)	NA	9124.73s

Table 3. Real times (seconds) for GCD problems.

In Table 3 column d is the maximum degree of the terms of $G, \bar{A}, \bar{B}$ in each variable, column t is the maximum number of terms of the coefficients of G. Timings are shown in seconds for the new algorithm for 1 core and 16 cores. For 1 core we show the %age of the time spent evaluating the inputs, that is computing $K(A)(x_0, \alpha^j)$ and $K(B)(x_0, \alpha^j)$ for $j = 1, 2, \ldots, T$. The parallel speedup on 16 cores is shown in parens.

Table 3 shows that most of the time in the new algorithm is in evaluation. It shows a parallel speedup approaching the maximum of 11.7 on this machine. There was a parallel bottleneck in how we computed the $\lambda_i(z)$ polynomials that limited parallel speedup to 10 on these benchmarks. For N cores, after generating a new batch of N images we used the Euclidean algorithm for Step 12b which is quadratic in the number of images j computed so far. To address this we now use an incremental version of the Berlekamp-Massey algorithm which is $O(Nj)$.

6.2 Benchmark 2

Our second benchmark (see Table 4) is for 9 variables where the degree of $G, \bar{A}, \bar{B}$ is at most 20 in each variable. The terms are generated at random as before but restricted to have total degree at most 60. The row with $\#G = 10^4$ and $\#A = 10^6$ is our benchmark problem from Section 1. We show two sets of timings for our new algorithm. The first set is for projecting down to univariate image GCDs in $\mathbb{Z}_p[x_0]$ and the second set it for bivariate GCDs and consequently the values of t are different.

The timings for the new algorithm are for the first prime only. Although one prime is sufficient for these problems to recover H that is, no Chinese remaindering is needed, our algorithm uses an additional 63 bit prime to verify $H \mod p_1 = H$. The time for the second prime is always less than 50% of the time for the first prime because it needs only $t + 1$ points instead of $2t + \delta$ points and it does not need to compute degree bounds.

For $\#G = 10^3$, $\#A = 10^5$, the time of 497.2s breaks down as follows. 38.2s was spent in computing degree bounds for G, 451.2s was spent in evaluation, of which 43.2s was spent computing the powers. Using the support of H from this first prime it took 220.9s to compute H modulo a second prime.

Table 4 shows again that most of the time in the new algorithm is in evaluation. This is also true of Zippel's algorithm and hence of Maple and Magma too. Because Maple uses random evaluation points, and not a power sequence, the cost of each evaluation in Maple is $O(n(\#A + \#B))$ multiplications instead of $\#A + \#B$ evaluations for the new algorithm. Also, Maple is using `% p` to divide in C which generates a hardware division instruction which is much more expensive than a hardware multiplication instruction. For the new algorithm, we are using Roman Pearce's implementation of Möller and Granlund [Moller and Granlund 2011] which reduces division by p to two multiplications plus other cheap operations. Magma is doing something similar. It is using floating point primes (25 bits) so that it can multiply modulo p using floating point multiplications. This is one reason shy Maple is slower than Magma.

In comparing the new algorithm with Maple's implementation of Zippel's algorithm, for $n = 8, d = 50$ in Table 3 we achieve a speedup of a factor of 1815 = 6390.22/3.52 on 1 core. Since Zippel's algorithm uses $O(dt)$ points and our Ben-Or/Tiwari algorithm uses $2t + O(1)$ points, we get a factor of $O(d)$ speedup because of this.

		New GCD: univariate images					New GCD: bivariate images					Zippel's algorithm	
#G	#A	t	1 core	(eval)	16 cores		t	1 core	(eval)	16 cores		Maple	Magma
10^2	10^5	13	0.14s	(62%)	0.043	3.3x	4	0.156s	(65%)	0.032s	2.8x	51.8s	10.92s
10^3	10^5	113	0.59s	(66%)	0.100s	5.9x	13	0.306s	(55%)	0.066s	4.6x	210.9s	60.24s
10^4	10^5	1197	7.32s	(48%)	1.022s	7.2x	118	2.299s	(36%)	0.224s	10.3x	7003.4s	10.84s
10^2	10^6	13	1.36s	(70%)	0.167s	8.2x	4	1.024s	(60%)	0.164s	6.2x	797.4s	45.08s
10^3	10^6	130	5.70s	(90%)	0.520s	11.0x	14	1.713s	(69%)	0.228s	7.1x	2135.9s	207.63s
10^4	10^6	1198	48.17s	(87%)	4.673s	10.3x	122	7.614s	(75%)	0.685s	11.1x	22111.6s	1611.46s
10^5	10^6	11872	466.09s	(82%)	45.83s	10.2x	1115	80.04s	(57%)	6.079s	13.2x	NA	876.89s
10^2	10^7	11	12.37s	(67%)	1.46s	8.5x	3	10.69s	(58%)	1.63s	6.6x	NA	354.90s
10^3	10^7	122	47.72s	(91%)	4.470s	10.7x	16	15.76s	(71%)	2.09s	7.5x	NA	1553.91s
10^4	10^7	1212	429.61s	(98%)	37.72s	11.4x	122	57.23s	(90%)	5.10s	11.2x	NA	8334.93s
10^5	10^7	11867	3705.4s	(98%)	311.6s	11.9x	1114	438.87s	(90%)	34.4s	12.8x	NA	72341.0s
10^6	10^7	117508	47568.s	(90%)	3835.9s	12.4x	11002	4794.5s	(83%)	346.1s	13.8x	NA	NA
10^2	10^8	12	129.26s	(69%)	15.86s	8.2x	4	101.8s	(60%)	16.88s	6.0x	NA	NA
10^3	10^8	121	522.14s	(92%)	49.17s	10.6x	17	150.0s	(73%)	23.25s	6.4x	NA	NA
10^4	10^8	1184	4295.0s	(99%)	412.69s	10.4x	121	555.5s	(89%)	78.76s	7.0x	NA	NA
10^5	10^8	11869	43551.s	(99%)	3804.93s	11.4x	1162	4417.7s	(98%)	626.19s	7.0x	NA	NA

Table 4. Timings (seconds) for 9 variable GCDs

7 CONCLUSION AND FINAL REMARKS

We have shown that a Kronecker substitution can be used to reduce a multivariate GCD computation to bivariate by using a discrete logs Ben-Or/Tiwari point sequence. Our parallel algorithm is fast and practical. For polynomials in more variables or higher degree algorithm PGCD may need a prime p larger than what a 63 bit prime for a 64 bit machine. Can we do anything to reduce the size of the prime needed?

We cite the sparse interpolation methods of [Garg and Schost 2009], [Giesbrecht and Roche 2011] and Roche [Arnold et al. 2016] which can use a smaller prime and would also use fewer than $2t + O(1)$ evaluations. These methods compute $a_i = K_r(A)(x, y)$, $b_i = K_r(B)(x, y)$ and $g_i = \gcd(a_i, b_i)$ all mod $\langle p, y^{q_i} - 1\rangle$ for several primes q_i and recover the exponents of y in $K_r(H)$ using Chinese remaindering. The algorithms differ in the size of q_i and how they avoid and recover from exponent collisions modulo q_i. It is not clear whether this approach can work for the GCD problem as these methods assume a division free evaluation but computing g_i modulo $\langle p, y^{q_i-1}\rangle$ requires division and $y = 1$ may be bad or unlucky. These methods also require $q_i \gg t$ which means computing g_i modulo $\langle p, y^{q_i} - 1\rangle$ will be expensive for large t. Instead of pursuing this direction we chose to implement a 127 bit prime version of our algorithm which proved to be not difficult. A 127 bit prime will cover almost all multivariate GCD problems arising in practice.

8 ACKNOWLEDGEMENT

We would like to acknowledge Adriano Arce for his discrete logarithm code, Alex Fan for his Chinese remaindering code, Alan Wong for integrating the bivariate GCD code and Roman Pearce for his 64 bit and 128 bit modular multiplication codes.

9 NOTE FOR THE REFEREES

An early version of this paper was presented at and published in the proceedings of ISSAC 2016. It's available in the ACM Digital Library at https://dl.acm.org/citation.cfm?id=2930903

In that paper we presented a version of our GCD algorithm, a heuristic version, for computing the GCD modulo the first prime p. One of the referees asked for a Las Vegas version of our algorithm with a complete analysis for the probability of failure. We were unable to do that at the time.

The version of our algorithm in Section 3 Simplified Algorithm is the Las Vegas algorithm that the referee asked for. So Section 3 is new. In addition, in Section 4 Faster Algorithm we redesigned our heuristic algorithm so that we can give a more formal and complete proof of termination. Sections 4.1-4.4 are new. The practical improvements in sections 5.2 and 5.3 are also new. The timings for Benchmark 2 in Section 6.2 are also new. They include the practical improvements we made in sections 5.2 and 5.3 and timings for newer versions of Maple and Magma.

Finally, in the 2016 paper we didn't have any space to give a treatment for the Chinese remaindering. The algorithms in the new paper include Chinese remaindering (in subroutines MGCD and MGCD1) to complete the GCD algorithm. For this purpose Proposition 2.9 and Theorem 2.10 in Section 2 are also new.

REFERENCES

Andrew Arnold, Mark Giesbrecht, and Daniel S. Roche. 2016. Faster sparse multivariate polynomial interpolation of straight-line programs. *J. Symb. Comp.* 75 (2016), 4–24.

Nadia B. Atti, Gema M. Diaz-Toca, and Henri Lombardi. 2006. The Berlekamp-Massey algorithm revisited. *Applicable Algebra in Engineering, Communication and Computing* 17 (April 2006), 75–82.

Michael Ben-Or and Prasoon Tiwari. 1988. A deterministic algorithm for sparse multivariate polynomial interpolation. In *Proceedings of STOC '88*. ACM, New York, NY, 301–309. https://doi.org/10.1145/62212.62241

Elwyn Berlekamp. 1970. Factoring polynomials over large finite fields. *Math. Comp.* 24 (1970), 713–735.

William S. Brown. 1971. On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors. *J. ACM* 18 (October 1971), 478–504. https://doi.org/10.1145/321662.321664

William S. Brown and Joseph F. Traub. 1971. On Euclid's Algorithm and the Theory of Subresultants. *J. ACM* 18 (October 1971), 505–514. https://doi.org/10.1145/321662.321665

Bruce W. Char, Keith O. Geddes, and Gaston H. Gonnet. 1989. GCDHEU: heuristic polynomial GCD algorithm based on integer GCD computation. *J. Symb. Comp.* 7 (1989), 31–48.

George E. Collins. 1967. Subresultants and reduced polynomial remainder sequences. *J. ACM* 14 (January 1967), 128–142. https://doi.org/10.1145/321371.321381

David A. Cox, John Little, and Donal O'Shea. 1991. *Ideals, Varities and Algorithms*. Springer-Verlag, New York, NY.

Sanchit Garg and Eric Schost. 2009. Interpolation of polynomials given by straight-line programs. *Theoretical Computer Science* 410, 27-29 (2009), 2659–2662.

K. O. Geddes, S. R. Czapor, and G. Labahn. 1992. *Algorithms for Computer Algebra*. Kluwer Academic Publishers, Boston, MA.

A. O. Gelfond. 1952. *Transcendental and Algebraic Numbers*. GITTL, Moscow. English translation by Leo F. Boron, Dover, New York, 1960.

Mark Giesbrecht and Daniel S. Roche. 2011. Diversification improves interpolation. In *Proceedings of ISSAC 2011*. ACM Press, 123–130. https://doi.org/10.1145/1993886.1993909

A. J. Goldstein and R. L. Graham. 1974. A Hadamard-type bound on the coefficients of a determinant of polynomials. *SIAM Rev.* 16, 3 (1974), 394–395.

Mahdi Javadi and Michael Monagan. 2010. Parallel Sparse Polynomial Interpolation over Finite Fields. In *Proceedings of PASCO 2010*. ACM Press, 160–168. https://doi.org/10.1145/1837210.1837233

E. Kaltofen, Y.N. Lakshman, and J-M. Wiley. 1990. Modular Rational Sparse Multivariate Interpolation Algorithm. In *Proceedings of ISSAC 1990*. ACM Press, 135–139. https://doi.org/10.1145/96877.96912

Erich Kaltofen, Wen shin Lee, and Austin A. Lobo. 2000. Early Termination in Ben-Or/Tiwari Sparse Interpolation and a Hybrid of Zippel's algorithm. In *Proceedings of ISSAC 2000*. ACM Press, 192–201. https://doi.org/10.1145/345542.345629

Erich Kaltofen, Wen shin Lee, and Austin A. Lobo. 2010. Fifteen years after DSC and WLSS2. what parallel computations I do today. In *Proceedings of PASCO 2010*. ACM Press, 10–17. https://doi.org/10.1145/1837210.1837213

Erich Kaltofen and Barry Trager. 1990. Computing with polynomials given by black boxesfor their evaluations: greatest common divisors, factorization, separation of numerators and denominators. *J. Symb. Comp.* 9 (March 1990), 301–320–.

J. L. Massey. 1969. Shift-register synthesis and BCH decoding. *IEEE Transactions on Information Theory* 15 (January 1969), 122–127.

Niels Moller and Torbjorn Granlund. 2011. Improved division by invariant integers. *IEEE Trans. Comput.* 60, 2 (February 2011), 165–175.

Michael Monagan and Alan Wong. 2017. Fast parallel multi-point evaluation of sparse polynomials. In *Proceedings of PASCO 2017*. ACM digital library. https://doi.org/10.1145/3115936.3115940

Gary Mullen and Daniel Panario. 2013. *Handbook of Finite Fields*. CRC Press.

Hirokazu Murao and Tetsuro Fujise. 1996. Modular Algorithm for Sparse Multivariate Polynomial Interpolation and its Parallel Implementation. *J. Symb. Comp.* 21, 4-6 (1996), 377–396.

S. Pohlig and M. Hellman. 1978. An improved algorithm for computing logarithms over GF(p) and its cryptographic significanc. *IEEE Transactions on Information Theory* 24, 1 (January 1978), 106–110.

Mohamed Omar Rayes, Paul S. Wang, and Kenneth Weber. 1994. Parallelization fo the Sparse Modular GCD Algorithm for Multivariate Polynomials on Shared Memory Processors. In *Proceedings of ISSAC '94*. ACM Press, 66–73. https://doi.org/10.1145/190347.190368

Jack Schwartz. 1980. Fast probabilistic algorithms for verification of polynomial identities. *J. ACM* 27, 4 (October 1980), 701–717. https://doi.org/10.1145/322217.322225

Douglas Stinson. 2006. *Cryptography, Theory and Practice*. Chapman and Hall.

Y. Sugiyama, M. Kashara, S. Hirashawa, and T. Namekawa. 1975. A Method for Solving Key Equation for Decoding Goppa Codes. *Information and Control* 27 (January 1975), 87–99.

Joris van der Hoven and Grégoire Lecerf. 2013. On the bit complexity of sparse polynomial multiplication. *J. Symb. Comp.* 50 (2013), 227–254.

Joris van der Hoven and Grégoire Lecerf. 2014. Sparse polynomial interpolation in practice. *ACM Communications in Computer Algebra* 48 (September 2014), 187–191. https://doi.org/10.1145/2733693.2733721

Joachim von zur Gathen and Jurgen Gerhard. 1999. *Modern computer algebra*. Cambridge University Press, New York, NY.

Paul Wang. 1980. The EEZ-GCD algorithm. *ACM SIGSAM Bulletin* 14 (May 1980), 50–60. https://doi.org/10.1145/1089220.1089228

Paul S. Wang. 1978. An Improved Multivariate Polynomial Factoring Algorithm. *Math. Comp.* 32, 144 (October 1978), 1215–1231.

Richard Zippel. 1979. Probabilistic algorithms for sparse polynomials. In *Proceedings of EUROSAM '79*. ACM, 216–226.

Richard Zippel. 1990. Interpolating Polynomials from their Values. *J. Symb. Comp.* 9, 3 (1990), 375–403.

Appendix

Maple code for the 6 variable gcd benchmark.

```
r := rand(2^31);
X := [u,v,w,x,y,z];
getpoly := proc(X,t,d) local i,e;
     e := rand(0..d);
     add( r()*mul(x^e(),x=X), i=1..t );
end:

infolevel[gcd] := 3; # to see output from Zippel's algorithm

for d in [5,10,20,50,100] do
  s := 100; t := 100*d;
  g := add(x^d,x=X) + r() + getpoly(X,t-7,d-1);
  abar := getpoly(X,s-1,d) + r();  a := expand(g*abar);
  bbar := getpoly(X,s-1,d) + r(); b := expand(g*bbar);
  st := time(); h := gcd(a,b); gcdtime := time()-st;
  printf("d=%d  time=%8.3f\n",d,gcdtime);
end do:
```

Magma code for the 6 variable gcd benchmark.

```
p := 2^31;
Z := IntegerRing();
P<u,v,w,x,y,z> := PolynomialRing(Z,6);
```

```
randpoly := function(d,t)
M := [ u^Random(0,d)*v^Random(0,d)*w^Random(0,d)
      *x^Random(0,d)*y^Random(0,d)*z^Random(0,d) : i in [1..t] ];
C := [ Random(p) : i in [1..t] ];
g := Polynomial(C,M);
return g;
end function;

for d in [5,10,20,50] do
  s := 100; t := 100*d;
  g := u^d+v^d+w^d+x^d+y^d+z^d + randpoly(d,t-7) + Random(p);
  abar := randpoly(d+1,s-1) + Random(p);  a := g*abar;
  bbar := randpoly(d+1,s-1) + Random(p);  b := g*bbar;
  d;  time h := Gcd(a,b);
end for;
```