On the Genericity of the Modular Polynomial GCD Algorithm*

Erich Kaltofen
North Carolina State University
Mathematics Department, Box 8205
Raleigh, N.C. 27695-8205 USA.
E-mail: `kaltofen@math.ncsu.edu`
URL: `www.math.ncsu.edu/~kaltofen`

Michael Monagan
Centre for Experimental and Constructive Mathematics
Simon Fraser University
Burnaby, British Columbia, V5A 1S6 Canada.
E-mail: `monagan@cecm.sfu.ca`

Abstract

In this paper we study the generic setting of the modular GCD algorithm. We develop the algorithm for multivariate polynomials over Euclidean domains which have a special kind of remainder function. Details for the parameterization and generic Maple code are given. Applying this generic algorithm to a GCD problem in $\mathbb{Z}/(p)[t][x]$ where p is small yields an improved asymptotic performance over the usual approach, and a very practical algorithm for polynomials over small finite fields.

1 Introduction

Efficient computation of greatest common divisors (GCDs) of multivariate polynomials is very important in computer algebra systems because the GCD operation is the bottleneck of many basic applications. The modular GCD algorithm, developed by Brown [2], solved the problem of efficient GCD computation in $\mathbb{Z}[x_1, ..., x_n]$, where $\mathbb{Z}$ denotes the integers, when the input polynomials are dense. However, Brown's algorithm is not effective when the polynomials are sparse. This problem was solved by Zippel [24, 25] in 1979. Many other methods have also been developed for efficient GCD computation in $\mathbb{Z}[x_1, \ldots, x_n]$. It is possible to compute the GCD of two polynomials viewed a black boxes for their evaluation again as a black box [17, 5]. Furthermore, Hensel lifting can be utilized in what is called the EZ-GCD algorithm (see [13, section 7] and the papers by Moses, Wang, and Yun cited there). Brown's dense modular algorithm, the EZ-GCD algorithm, and Zippel's sparse modular algorithm are all widely used in practice. The modular algorithms can be readily parallelized [22].

In this paper we investigate the general setting of the modular GCD algorithm. In Section 2 we outline the modular algorithm in some detail for reference later in the paper. In Section 3 we show how it can be applied to GCD problems over the Euclidean ring $\mathbb{Z}/(p)[t]$, where $\mathbb{Z}/(p)$ denotes the integer residues modulo p. We compare it with other algorithms in the domain $\mathbb{Z}/(p)[t][x]$ and establish it as better than the known other standard methods. In section 4 we investigate the general setting of the modular algorithm. Maple code for a generic implementation of the modular GCD algorithm is given there, along with implementation notes.

*This material is based on work supported in part by the National Science Foundation under Grant No. CCR-9712267 (Erich Kaltofen) and on work supported by NSERC of Canada (Michael Monagan). Version 3/31/1999 (12:10).

2 The Modular GCD Algorithm

In this section we outline the modular GCD algorithm for multivariate polynomials with integer coefficients. A description of the algorithm may be found in section 7.4 in [10]. Throughout we assume that all GCDs that are computed are to be unit normal.

Let $P = \mathbb{Z}[x_1, \ldots, x_n]$, $p \in \mathbb{Z}$ be a prime and $G = \mathbb{Z}/(p)[x_1, \ldots, x_n]$. Let $a, b \in P\setminus\{0\}$, $g = \mathrm{GCD}(a, b)$, $a = g\bar{a}$, and $b = g\bar{b}$. The modular algorithm computes the GCD g in two stages. The first stage applies a sequence of modular homomorphisms $\phi_p\colon P \to G$ where $\phi_p(a) = a \bmod p$ to reduce the GCD problem to a sequence of GCD problems in $\mathbb{Z}/(p)[x_1, \ldots, x_n]$. The GCD g is reconstructed from the solution of these GCD problems by application of the Chinese remainder theorem. This first stage of the algorithm is presented here for later reference. The GCD computations occurring in step 9 are calculated in the second stage which is described later.

Algorithm ModularGCD Over the Integers

Inputs: $A, B \in P = \mathbb{Z}[X] \setminus \{0\}$ where $X = x_1, \ldots, x_n$.
Output: $g = \mathrm{GCD}(A, B)$.

1 $ca \leftarrow \mathrm{content}_P(A)$.
2 $cb \leftarrow \mathrm{content}_P(B)$.
3 $c \leftarrow \mathrm{GCD}_{\mathbb{Z}}(ca, cb)$.
4 $(a, b) \leftarrow (A/ca, B/cb)$.
5 $\gamma \leftarrow \mathrm{GCD}_{\mathbb{Z}}(\mathrm{lc}_X(a), \mathrm{lc}_X(b))$,
where lc is the leading coefficient (see below).
6 $\mathbf{d} \leftarrow \min_X(\deg(a), \deg(b))$,
where deg and min use a term ordering.
7 $(m, g_m) \leftarrow (1, 0)$.
8 Loop: Find a prime $p \in \mathbb{Z}$ such that $p \nmid m\gamma$.
9 $g_p \leftarrow \mathrm{GCD}_G(\phi_p(a), \phi_p(b))$.
10 If $\deg(g_p) =_X \mathbf{0}$ then output c and stop.
11 If $\deg(g_p) >_X \mathbf{d}$ then Goto step 8.

12 $\quad g_p \leftarrow (\gamma \bmod p)/\operatorname{lc}_X(g_p)\cdot g_p$.
13 $\quad$ If $\deg(g_p) <_X \mathbf{d}$ then $(m, g_m, \mathbf{d}) \leftarrow (1, 0, \deg(g_p))$.
14 $\quad h \leftarrow \mathrm{CRA}([g_p, p], [g_m, m])$.
15 $\quad$ If $h = g_m$ then begin termination test.
16 $\quad\quad h \leftarrow h/\operatorname{content}_P(h)$.
17 $\quad\quad$ If $h \mid a$ and $h \mid b$ then output $c\cdot h$ and stop.
18 $\quad (m, g_m) \leftarrow (p\cdot m, h)$.
19 Goto step 8.

Note, the result computed by the Chinese remainder algorithm (CRA) in step 13 must be expressed in the symmetric range of the integers modulo $p\cdot m$ so that negative integer coefficients in g can be reconstructed as well as positive ones. We will come back to this later when we try to generalize the algorithm.

The basic elements of the proof of correctness of the algorithm are reproduced here for discussion later when we look at the general setting of the algorithm. The requirement that p not divide γ in step (8) means that Lemma 1 below holds. Lemma 1 tells us the relationship between the image GCD g_p that we compute in step (9) and what we want to compute, namely $\phi_p(g)$. The function $\operatorname{lc}_{X,\prec}(g)$ denotes the leading coefficient of the polynomial $g \in P$ under an admissible term ordering $\prec$ on the variables $X = x_1, x_2, \ldots, x_n$. We have not before and will not list the term ordering explicitly in the leading coefficient function. Also $\operatorname{lc}(a\cdot b) = \operatorname{lc}(a)\cdot\operatorname{lc}(b)$ will hold since $\mathbb{Z}$ is an integral domain.

Lemma 1 *Let $a, b \in \mathbb{Z}[x_1, \ldots, x_n]\setminus\{0\}$ be primitive. Let $g = \mathrm{GCD}(a, b)$ and let $g_p = \mathrm{GCD}(\phi_p(a), \phi_p(b))$. Let $\gamma = \mathrm{GCD}_{\mathbb{Z}}(\operatorname{lc}(a), \operatorname{lc}(b))$. If $\phi_p(\gamma) \neq 0$ then $g_p = \phi_p(g)\cdot\delta\cdot\Delta$ where $\delta \in \mathbb{Z}/(p)\setminus\{0\}$ and $\Delta \in G\setminus\{0\}$. Thus $\deg(g_p) \geq_X \deg(g)$.*

Proof. Let $a = g\bar{a}$ and $b = g\bar{b}$. Then $\mathrm{GCD}(\phi_p(a), \phi_p(b)) = \mathrm{GCD}(\phi_p(g\bar{a}), \phi_p(g\bar{b})) = \mathrm{GCD}(\phi_p(g)\cdot\phi_p(\bar{a}), \phi_p(g)\cdot\phi_p(\bar{b}))$. Since $\phi_p(\operatorname{lc}_X(g)) \neq 0$ then $\deg(\phi_p(g)) =_X \deg(g)$. Since a and b are primitive then $\phi_p(\bar{a})\phi_p(\bar{b}) \neq 0$ hence $\mathrm{GCD}(\phi_p(\bar{a}), \phi_p(\bar{b})) \neq 0$. Hence $g_p = \phi_p(g)/\operatorname{lc}_X(\phi_p(g))\cdot\mathrm{GCD}(\phi_p(\bar{a}), \phi_p(\bar{b})) = \phi_p(g)\cdot\delta\cdot\Delta$ as claimed. ⊠

A consequence of Lemma 1 is that if $\deg(g_p) =_X \mathbf{0}$ then $\deg(g) =_X \mathbf{0}$ hence the early termination of the algorithm in step 10.

If $\deg(\Delta) >_X \mathbf{0}$ then we say p is an unlucky prime. Conversely, if $\deg(\Delta) =_X \mathbf{0}$ we say that p is a good prime. The modular algorithm detects that the current prime p is an unlucky prime in step 11, and that all previous primes are unlucky in 13. The division checks $h \mid a$ and $h \mid b$ in step 17 detect any unlucky primes not caught in steps 11 or 13.

In the univariate case the unlucky primes are characterized by those which divide $r = \operatorname{res}_x(\bar{a}, \bar{b})$. Since r is an integer of finite size, the number of unlucky primes is finite. Lemma 2 below establishes that the number of unlucky primes is finite for the multivariate case.

Lemma 2 *The number of unlucky primes is finite.*

Proof. For later reference, we prove this Lemma for coefficients in a unique factorization domain U in place of $\mathbb{Z}$, and we shall count as one two prime elements p_1 and p_2 that are associates, i.e., if their quotient p_1/p_2 is an invertible element (a unit) in U. Note that in a UFD such as $\mathbb{Q}[x]$, where $\mathbb{Q}$ are the rational numbers, the number of units is infinite.

We first note a lemma by Gauss (see [9, Article 42], [18]) that relates a factorization in $\mathrm{QF}(U)[x_1, \ldots, x_n]$, where $\mathrm{QF}(U)$ is the field of quotients of U, to one in $U[x_1, \ldots, x_n]$. Suppose $A = a_1 \cdots a_r$, where $A \in U[x_1, \ldots, x_n]$ and $a_1, \ldots, a_r \in \mathrm{QF}(U)[x_1, \ldots, x_n]$. Then there exist $u_1, \ldots, u_r \in \mathrm{QF}(U)$ such that $A = (u_1 a_1)\cdots(u_r a_r)$ and $(u_1 a_1), \ldots, (u_r a_r) \in U[x_1, \ldots, x_n]$. Gauss's lemma actually justifies the above algorithm.

A prime p is unlucky exactly if $\mathrm{GCD}(\phi_p(\bar{a}), \phi_p(\bar{b})) \neq 1$, where $\bar{a}$ and $\bar{b}$ are the co-factors of a and b defined above. Since $\mathrm{GCD}(\bar{a}, \bar{b}) = 1$ in $U[x_1, \ldots, x_n]$, we have, by applying Gauss's lemma, $\mathrm{GCD}_{\mathrm{QF}(U)[x_1,\ldots,x_n]}(\bar{a}, \bar{b}) = 1$. Over the field $\mathrm{QF}(U)$ the GCD problem is a rational problem, which means that the GCD can be computed by arithmetic operations in $\mathrm{QF}(U)$ alone. Therefore we may extend the coefficient field by transcendental elements $z_2, \ldots, z_n$, thus proving that $\mathrm{GCD}_D(\bar{a}, \bar{b}) = 1$ for $D = \mathrm{QF}(U)(z_2, \ldots, z_n)[x_1, \ldots, x_n]$.

We now consider the map $\Psi\colon D \to E = \mathrm{QF}(U)(z_2, \ldots, z_n)[y_1, \ldots, y_n]$ defined by $\Psi(f(x_1, \ldots, x_n)) = f(y_1, y_2 + z_2 y_1, \ldots, y_n + z_n y_1)$ (cf. [15, Proof of Lemma 6.2]). Since Ψ is a polynomial ring isomorphism over the coefficient field $\mathrm{QF}(U)(z_2, \ldots, z_n)$ we must have

$$\mathrm{GCD}(\Psi(\bar{a}), \Psi(\bar{b})) = 1 \tag{1}$$

either in E or more general by Gauss's lemma in $\mathrm{QF}(U)(z_2, \ldots, z_n, y_2, \ldots, y_n)[y_1]$.

We now can consider the resultant

$$\varrho = \operatorname{res}_{y_1}(\Psi(\bar{a}), \Psi(\bar{b})) \in U[z_2, \ldots, z_n, y_2, \ldots, y_n].$$

From (1) it follows as a property of the resultant that $\varrho \neq 0$. The domain homomorphism $\phi_p\colon U \to U/(p)$ extends to the polynomial rings $U[z_2, \ldots, z_n, x_1, \ldots, x_n]$ and $U[z_2, \ldots, z_n, y_1, \ldots, y_n]$ and induces an isomorphism Ψ_p. Suppose that

$$\phi_p(\varrho) \neq 0. \tag{2}$$

Then $\mathrm{GCD}(\phi_p(\Psi(\bar{a})), \phi_p(\Psi(\bar{b}))) = 1$ in $U/(p)(z_2, \ldots, z_n, y_2, \ldots, y_n)[y_1]$. We now claim that there can be no common factor in $U/(p)(z_2, \ldots, z_n)[y_2, \ldots, y_n]$. First, not both of the leading coefficients of $\Psi(\bar{a})$ and $\Psi(\bar{b})$ with respect to y_1 can map to zero via ϕ_p, because otherwise their resultant ϱ would also map to zero. Suppose now, without loss of generality, that

$$\phi_p(\operatorname{lc}_{y_1}(\Psi(\bar{a}))) \neq 0. \tag{3}$$

The key property of the map Ψ is that $\operatorname{lc}_{y_1}(\Psi(\bar{a})) \in \mathrm{QF}(U)(z_2, \ldots, z_n)$, which yields from (3) the stronger condition $\mathrm{GCD}(\phi_p(\Psi(\bar{a})), \phi_p(\Psi(\bar{b}))) = 1$ in $U/(p)(z_2, \ldots, z_n)[y_1, y_2, \ldots, y_n]$. Using the isomorphism Ψ_p^{-1} with $y_1 = x_1$ and $y_i = x_i - z_i x_1$ for $2 \leq i \leq n$ we immediately obtain $\mathrm{GCD}(\phi_p(\bar{a}), \phi_p(\bar{b})) = 1$ in $U/(p)(z_2, \ldots, z_n)[x_1, \ldots, x_n]$. However, neither $\phi_p(\bar{a})$ nor $\phi_p(\bar{b})$ contain any z_i, so they are relatively prime in $U/(p)[x_1, \ldots, x_n]$.

Finally, we observe that all $p \in U$ that satisfy (2) must divide all coefficients in U of ϱ, thus their greatest common divisor which has a finite number of prime factors. ⊠

We give this proof, which is based on a single resultant, as it allows the sharpest estimate on the number and size of unlucky primes, especially when combined with the inequalities in [11]. However, we shall not analyze unluckiness further.

Assuming p is a good prime, that is, $\Delta = 1$, the image GCD $g_p = \mathrm{GCD}(\phi_p(a), \phi_p(b))$, which is monic, will be an associate of $\phi_p(g)$, differing from $\phi_p(g)$ by the unit $\delta = \mathrm{lc}_X(g) \bmod p$. If we knew $\mathrm{lc}_X(g)$ then we could set $g_p \leftarrow \delta g_p$ and the CRA would, given enough primes, reconstruct g. Instead we compute a multiple of g by imposing γ as a leading coefficient. The GCD g is now recovered when we have reconstructed αg by calculating $\alpha = \mathrm{content}_P(h)$ in step 16 and dividing it out.

Termination is ensured because eventually we will have sufficient good primes to reconstruct g. Instead of testing whether $h \mid a$ and $h \mid b$ at each iteration, which would be expensive, the test is made only when h does not change from one iteration to the next.

The Second Stage

In the second stage of the modular GCD algorithm we compute a GCD in $\mathbb{Z}/(p)[x_1, \ldots, x_n]$. The evaluation homomorphism $\phi_{x_n \leftarrow \alpha}$ with $\alpha \in \mathbb{Z}/(p)$ is applied to $\mathbb{Z}/(p)[x_n][x_1, \ldots, x_{n-1}]$. For the algorithm to terminate p must be sufficiently large so that sufficiently many good evaluation points exist. This is usually not a problem in practice because the primes chosen in stage one will be the largest primes that fit in a computer word, typically 31 or 63 bits in length. The evaluation homomorphism algorithm is applied recursively until we have a GCD problem in one variable, x_1, over $\mathbb{Z}/(p)$ where the Euclidean algorithm is used. After sufficiently many image GCDs have been computed in $\mathbb{Z}/(p)[x_1, \ldots, x_{n-1}]$ the coefficients in x_n are interpolated from the images.

It is convenient for the purpose of our presentation of the modular algorithm to view the evaluation homomorphism $\phi(a) = a(\alpha)$ as equivalent to the modular homomorphism $\phi(a) = a \bmod (x - \alpha)$. Replacing the Euclidean coefficient domain $\mathbb{Z}$ in Algorithm ModularGCD by the Euclidean domain $\mathbb{Z}/(p)[x_n]$, that is, with inputs $a, b \in \mathbb{Z}/(p)[x_n][x_1, \ldots, x_{n-1}]$, this second stage of the modular GCD algorithm becomes equivalent to the first stage where the primes p chosen in step 8 are now the linear polynomials $x_n - \alpha$, and interpolation becomes equivalent to the CRA in step 14. Lemma 1 and Lemma 2 also hold.

3 The Modular Algorithm over $\mathbb{Z}/(p)[t]$

Consider a GCD problem in $R = \mathbb{Z}/(p)[x_1, ..., x_n]$. Such problems arise when factoring polynomials in R where the first step is to perform a square-free factorization of the polynomial to determine any repeated factors. A bottleneck of the entire factorization process may be the first GCD computed in the square-free factorization step.

For example, consider the polynomial P in figure 1, which in expanded form has 158 terms. It is a polynomial in $\mathbb{Z}/(2)[A, B, C, Z]$. To factor this polynomial the square-free factorization step calculates the $\mathrm{GCD}(P, \partial P/\partial A)$. Maple V Release 5 applies the subresultant algorithm because it is unable to apply the modular algorithm over $\mathbb{Z}/(2)$ because not enough good evaluation points exist. Because the subresultant algorithm blows up, Maple does not terminate. Most of the polynomial factorization problems listed in [1] that arose in practice are polynomials over small finite fields, often $p = 2$.

What do we do if p is too small for the modular GCD algorithm to be applied because there are too few good evaluation points in $\mathbb{Z}/(p)$? One known solution to this problem is to choose evaluation points from $\mathrm{GF}(p^k)$, a field extension of $\mathbb{Z}/(p)$ where k is sufficiently large. To interpolate a polynomial of degree n we require $k \geq \log_p(1 + n)$.

```
z := (A^2*B^2*Z^2+A^2*B^2+B^2*Z^2+Z^2+B^2)/Z:
a := (B^2*C^2*A^2+B^2*C^2+C^2*A^2+A^2+C^2)/A:
b := (C^2*Z*B^2+C^2*Z+Z*B^2+B^2+Z)/B:
c := (Z*A*C^2+Z*A+A*C^2+C^2+A)/C:
P := Z^2*A^3*B^2*C^2*(z^2*b^2*c^2
                   +(a+z+1)^2*(a+c^2+1)):
P := expand(P) mod 2:
```

Figure 1: Maple code for H. Dobbertin's [7] polynomial

An alternative approach which we propose here is to consider $R = \mathbb{Z}/(p)[x_n][x_2, ..., x_{n-1}]$ and apply the modular algorithm to the coefficient ring $\mathbb{Z}/(p)[x_n]$ which is a Euclidean domain (see also [21, section 2.7.2]). Let m_i, $i = 1, 2, ...$ be a sequence of monic irreducible polynomials in $\mathbb{Z}/(p)[x_n]$. It is not difficult to see that the modular algorithm can be applied to compute a GCD in R by computing image GCDs modulo the m_i, that is over the finite fields $\mathbb{Z}/(p)[x_n]/(m_i)$, and applying the Chinese remaindering to the resulting images. Indeed in our presentation of the second stage of the modular GCD algorithm we did exactly this with linear polynomials $p = x_n - \alpha$.

The advantage of this approach over computing in an extension field is now demonstrated for the case of $R = \mathbb{Z}/(p)[t][x]$, i.e. bivariate polynomials. Let $\bar{a}, \bar{b}, g \in R$ where $\mathrm{GCD}(\bar{a}, \bar{b}) = 1$ and $\deg_t(\bar{a}) = \deg_t(\bar{b}) = \deg_t(g) = \deg_x(\bar{a}) = \deg_x(\bar{b}) = \deg_x(\bar{g}) = n$. Consider the problem of computing the $\mathrm{GCD}(a = \bar{a}g, b = \bar{b}g)$ in R. In the following complexity estimates we assume classical algorithms are used for polynomial arithmetic in $\mathbb{Z}/(p)[t]$ and R, i.e., multiplication and division are $O(n^2)$.

For comparison, we mention that the number of arithmetic operations in $\mathbb{Z}/(p)$ done by the subresultant algorithm with quadratic polynomial arithmetic is $O(n^6)$.

The approach of choosing evaluation points in $\mathrm{GF}(p^k)$, an extension field of $\mathbb{Z}/(p)$ requires that $k \geq \log_p(n+1)$ for there to be sufficiently many evaluation points. The cost of the modular evaluation algorithm when run over $\mathrm{GF}(p^k)$ will depend on how the finite field is represented. We assume polynomials of degree k are used to represent the field elements, and that multiplication and division in $\mathrm{GF}(p^k)$ are quadratic in k. Under these assumptions we have

Result 1 *The modular evaluation algorithm when run over* $\mathrm{GF}(p^k)$ *performs* $O(k^2 n^3)$ *arithmetic operations in* $\mathbb{Z}/(p)$.

Result 2 *The modular algorithm applied to* $\mathbb{Z}/(p)[t][x]$ *performs* $O(kn^3)$ *arithmetic operations in* $\mathbb{Z}/(p)$.

In both of the following analyses, the cost of the trial divisions in step 17, which are performed in $\mathbb{Z}/(p)[t][x]$, is not included. If the classical algorithm is used for trial division, the cost of the trial division step will be $O(n^4)$ arithmetic operations over $\mathbb{Z}/(p)$.* Furthermore, it is assumed that all primes are good.

*Alternatively and faster, namely in $O(n^3)$ arithmetic operations, the algorithms can interpolate/Chinese remainder the co-factors a and b. If the degrees of their coefficients stay properly bounded, i.e., if the maximum degree of the coefficients of each co-factor and the GCD adds up to the maximum degree of the coefficients of the corresponding input polynomial, the division is certified to be exact provided the moduli cover the degrees of the coefficients of the inputs.

Suppose the modular evaluation algorithm is used and the evaluation points are chosen from the field extension $K = \mathrm{GF}(p^k)$. The algorithm must compute $\mathrm{GCD}(a(\alpha_i,\, x), b(\alpha_i,\, x))$ at $O(n)$ points $\alpha_i \in \mathrm{GF}(p^k)$ in order to interpolate t of degree n in g. Each evaluation of $a(t,x)$ at $t = \alpha_i$ costs $O(n^2)$ arithmetic operations in $\mathrm{GF}(p^k)$. Each image GCD costs $O(n^2)$ arithmetic operations in $\mathrm{GF}(p^k)$. We must interpolate n coefficients in $\mathrm{GF}(p^k)[t]$ of degree n, each of which costs $O(n^2)$ arithmetic operations in $\mathrm{GF}(p^k)$. This is $O(n^3)$, $O(n^3)$ and $O(n^3)$ arithmetic operations in $\mathrm{GF}(p^k)$ for the evaluations, image GCDs, and interpolations, respectively. Thus we have $O(n^3k^2)$ arithmetic operations over $\mathbb{Z}/(p)$ assuming arithmetic in $\mathrm{GF}(p^k)$ is done using classical polynomial arithmetic.

Suppose the modular algorithm is applied to the coefficient ring $\mathbb{Z}/(p)[t]$. To reconstruct coefficient polynomials in $\mathbb{Z}/(p)[t]$ of degree n, we will require monic irreducibles $m_i(t) \in \mathbb{Z}/(p)[t]$ such that $\sum_i \deg_t(m_i) \geq n$. We first claim that there are sufficiently many irreducible polynomials such that we may choose $d_{\max} = \max_i(\deg_t(m_i)) = O(\log_p n)$. This claim is a direct consequence of

$$t^{p^k} - t = \prod_{\substack{f \text{ irreducible in } \mathbb{Z}/(p)[t] \\ \deg(f) \text{ divides } k}} f(t). \qquad (4)$$

The cost of each modular reduction requires division of $O(n)$ coefficients in $\mathbb{Z}/(p)[t]$ of degree n by $m_i(t)$ of degree $d_i = O(\log_p n)$. The total cost is $O(n\sum_i(nd_i)) = O(n^3)$ arithmetic operations over $\mathbb{Z}/(p)$, as we also can have $\sum_i d_i = O(n)$. The cost of each image GCD is $O(n^2)$ arithmetic operations over $\mathrm{GF}(p^{d_i})$. For all image GCDs this is $O(n^2 \sum_i d_i^2)$ arithmetic operations over $\mathbb{Z}/(p)$. Since $\sum_i d_i^2 \leq d_{\max} \sum_i d_i = O(n \log_p n)$ the cost of the image GCDs is $O(n^3k)$ for any $k > \log_p n$. There are $n+1$ coefficients to reconstruct using the CRA, each of which take $O(n^2)$ arithmetic operations over $\mathbb{Z}/(p)$. The $O(n^3k)$ arithmetic operations over $\mathbb{Z}/(p)$ in the image GCD computations therefore dominate the cost.

Thus there is an improvement of a factor of $k = O(\log_p n)$ over the "standard" approach. The reason for the improvement is that we are using t instead of a new variable to represent $\mathrm{GF}(p^k)$. Dan Grayson has offered us another view of this trick: evaluation in $\mathrm{GF}(p^k)$ yields k images of the GCD, all algebraic conjugates of the extension element. Our Chinese remaindering approach shows how to make use of these conjugates and recover the coefficients in $\mathbb{Z}/(p)$. However, Grayson's view is of value when the input polynomials are presented by black boxes [17, 6], as so-called extended domain black boxes can allow for evaluation in $\mathrm{GF}(p^k)$, but there would be no easy mechanism for modular reduction of the coefficients in a single variable.

In practice, we have observed a better gain than the expected factor of k on problems where $p = 2$ in Maple for the following reason. We seek in the following table to show the penalty that one pays when when one computes over $\mathrm{GF}(p^k)$ instead of $\mathbb{Z}/(p)$ using polynomials to represent $\mathrm{GF}(p^k)$. In Maple, polynomials in $\mathbb{Z}/(p)[t]$ are represented as arrays of machine integers and arithmetic is done in-place in compiled C code. Multiplication, division, GCD, are quadratic in k. See [20] for details of the Maple representation for $\mathbb{Z}/(p)[t]$. In comparison, polynomials in $\mathrm{GF}(p^k)[t]$ are represented as Maple lists of polynomials in $\mathbb{Z}/(p)[t]$. Arithmetic is not done in-place and is interpreted in Maple.

Let $\bar{a}, \bar{b}, g \in \mathrm{GF}(11^k)[x]$ where $\deg(\bar{a}) = \deg(\bar{b}) = \deg(g) = n$. The data in column 3 of Table 1 for $T(n,k)$ were generated by timing how long Maple V Release 5 takes to calculate $\mathrm{GCD}(a = \bar{a}g, b = \bar{b}g)$ for different values of k, for suitably chosen n, where the coefficients of the polynomials in $\mathrm{GF}(11^k)$ were generated at random.

k	n	$T(n,k)$	$\frac{T(n,k)}{T(n,1)}$	n	$M(n,k)$	$\frac{M(n,k)}{M(n,1)}$
2	256	.96e-4	242	24	.61e-5	15.6
4	196	.31e-4	79	21	.26e-5	6.5
8	144	.95e-5	24	18	.15e-5	3.7
16	100	.29e-5	7.4	15	.18e-5	4.5
32	64	.17e-5	4.3	12	.10e-5	2.6
64	36	.99e-6	2.5	9	.75e-6	1.9
128	16	.60e-6	1.5	6	.82e-6	2.1

Table 1: $n^2k^2T(n,k)$ is the time in seconds for a GCD over $\mathrm{GF}(11^k)$; $n^2k^2M(n,k)$ is the time in seconds for a GCD over $\mathbb{Z}/(p)$, where p is a prime k words (of memory) long.

The data in column 4 for $T(n,k)/T(n,1)$ indicates the penalty or loss of efficiency in calculating over $\mathrm{GF}(11^k)$ for $k > 1$ compared with $\mathbb{Z}/(11)$. The factor 242 for $k = 2$ means that there is an increase in cost of a factor of 242 beyond the expected factor of 4 from k^2. The factor 1.5 for $n = 128$ means that the penalty beyond the factor of 128^2 is now only 50%. There are two reasons for this. First, the Maple code for GCD computation in $\mathrm{GF}(p^k)[x]$ is interpreted (the coefficient arithmetic is compiled) and this overhead is relatively high when k is small whereas the code for GCD computation for $\mathbb{Z}/(p)[x]$ is compiled C code. Second, that whereas the data structure for polynomials in $\mathbb{Z}/(p)[x]$ is an array of machine integers, and the GCD computation runs "in-place" with no additional memory required, this is not the case for $\mathrm{GF}(11^k)[x]$ where the coefficient arithmetic requires data structures to be allocated as needed.

We have attempted in columns 6 and 7 to eliminate the overhead of the Maple interpreter from the picture by timing GCDs over $\mathbb{Z}/(p)$ instead of $\mathrm{GF}(11^k)$ where p is a k word prime. Again, the coefficients were generated at random. In this case the Maple code is compiled and runs in-place. We can see from column 7 that for small k there is still a penalty of a factor of 15. Thus, if one chooses to represent $\mathrm{GF}(p^k)$ using polynomials, there will always be a significant penalty.

We can take advantage of this in the modular GCD algorithm by increasing the degree of the irreducibles (with a negligible penalty) but requiring fewer irreducibles. We observed in our implementation that for $\mathbb{Z}/(p)[x]$ the optimal degree for the irreducibles is arround 12.

4 The Modular Algorithm over Euclidean Domains

The purpose of a generic algorithm is to present a single program that can be used in multiple settings. As we have seen in sections 2 and 3, the modular GCD algorithm is employed for the coefficient domains $\mathbb{Z}, \mathbb{Z}/(p)[x_i]$ and $\mathrm{GF}(p^k)[x_{i-1}]$ to produce GCDs over $\mathbb{Z}$ and $\mathbb{Z}/(p)$. An extension of Brown's algorithm to Gaussian integers is described in [4]. All these domains share to property of a Euclidean division, and we shall discuss the algorithm for a Euclidean coefficient domain.

By definition, a Euclidean domain E is an integral domain with a remainder function rem: $E \times (E \setminus \{0\}) \to E$, such that there exists a degree function $\delta\colon E \to \mathbb{Z}_{\geq 0}$ with the property that for any $a \in E$ and any $b \in E \setminus \{0\}$ there exists a quotient $q \in E$ and a remainder $r \in E$ with the properties

$$a = qb + r \text{ and } r = 0 \text{ or } (r \neq 0 \text{ and } \delta(r) < \delta(b)).$$

Any Euclidean domain is a unique factorization domain. This fact is easier to prove if one requires $\delta(ab) \geq \delta(a)$ for all $a, b \in E$ with $ab \neq 0$, but this is not necessary.

A prime in an integral domain is an non-zero element p with the property that $p \mid a \cdot b$ implies $p \mid a$ or $p \mid b$. An irreducible element $q \neq 0$ in an integral domain has the property that $q = u \cdot v$ implies that either u or v is a unit. In a unique factorization domain both notions are the same. The elements in the residue ring $E/(m)$ (often called the "quotient ring") are congruence classes modulo $m \in E$, which are normally represented by elements in E. The representation need not be canonical, i.e., two elements $a, b \in E$ may represent the same class. Such ambiguity is well-understood: for example, the two fractions 1/2 and 2/4 represent the same rational number. In order to perform ring arithmetic in $E/(m)$ we need a test for equality: $a \equiv b \bmod m$ if $m \mid a - b$. The alternative is canonical simplification, which is a procedure that reduces any representative of a congruence class of $E/(m)$ to an element unique to that class (see below). Often, Euclidean division by m is a canonical simplification. We use the notion $a \bmod m$ ambiguously: first, it denotes the congruence class in $E/(m)$ that contains a; second, it is a (possibly canonical) element in E that represents the same class.

In a Euclidean domain E a prime p must produce as a residue ring $E/(p)$ a field. We obtain an algorithm for the reciprocal of $(m \bmod p) \in E/(p)$, where $m \in E \setminus pE$ via the extended Euclidean algorithm: the algorithm computes $s, t \in E$ such that $sm + tp = 1$, so $s \bmod p$ is the reciprocal of $m \bmod p$. Chinese remaindering is thus facilitated: suppose we have $a \bmod m$ and $b \bmod p$ where $\mathrm{GCD}(m, p) = 1$. Then

$$c \equiv a + ((b - a)m^{-1} \bmod p) \cdot m \pmod{mp}$$

is a representative of a unique congruence class in $E/(mp)$ with $c \equiv a \pmod{m}$ and $c \equiv b \pmod{p}$.

In any integral domain association is an equivalence relation on all non-zero elements: a is an associate to b if b divides a and a/b is a unit. Our algorithm returns a unique GCD, namely that associate whose leading coefficient is unit normal. Unit normalization is a canonical simplification algorithm for association, which is part of the generic interface to the integral domain. Divisibility testing and exact division is another interface operation. Note that unit normalization need not be multiplicative (e.g., $\mathbb{Z}[i]$), which must be accounted for in step 17.

The generalization of the modular GCD algorithm presented in Section 2 to Euclidean domains imposes additional conditions on E. For one, we must have an infinite source of primes. We shall suppose that we have access to the stream of unassociated primes

$$p_1, p_2, p_3, \ldots \text{ with } \delta(p_1) < \delta(p_1 p_2) < \delta(p_1 p_2 p_3) \cdots. \quad (5)$$

A second issue is the recovery of elements a in E from $a \bmod m$ for sufficiently large m. Even over the integers, the divisibility test in step 17 will only succeed if the residues modulo m are chosen in the symmetric range $\{\lfloor -(m-1)/2 \rfloor, \ldots, -1, 0, 1, \ldots, \lceil -(m-1)/2 \rceil\}$. Similar difficulties are addressed be the landmark generic description of the Zassenhaus-Hensel algorithm in [21]. Instead of the requirement of an exhaustive set of representatives like in [21, section 2.6.2], we impose the condition on a required canonical simplifier modulo m. Recall that $\chi\colon E \times (E \setminus \{0\}) \to E$ is a canonical simplifier for the residue domain construction if

Simplification for all $a \in E$ we have $a \equiv \chi(a, m) \pmod{m}$, and

Canonicity for all $a, b \in E$ with $a \equiv b \pmod{m}$ we have $\chi(a, m) = \chi(b, m)$.

Note that $\chi(\chi(a, m), m) = \chi(a, m)$. The required condition is hard to read, at first:

$$\forall B > 0\, \exists M(B) \text{ such that } \forall m \text{ with } \delta(m) \geq M(B) \text{ and } \forall a \text{ with } \delta(a) < B\colon \chi(a, m) = a. \quad (6)$$

In words, all elements of a bounded degree are recovered by the simplifier if the modulus is sufficiently large. Here is how the condition works. The bound B is a degree bound with respect to the degree function δ for the coefficients of $\gamma/\mathrm{lc}(g)g$, which is our target polynomial for h. If $\delta(m) \geq M(B)$ then $\chi(h, m) = \chi(g_m, m) = \gamma/\mathrm{lc}(g)g$ in step 15. For $E = \mathbb{Z}$ we may choose $M(B) = 2B$ and χ symmetric remaindering modulo m, for $\mathbb{Z}/(p)[t]$ we may choose $M(B) = B$ and χ normal polynomial remaindering. The condition (5) on the primes ensures that $\delta(m)$ will reach $M(B)$.

Finally, we call a Euclidean remainder function a symmetric remainder if it simultaneously is a canonical simplifier with property (6). It is, however, not advisable to employ symmetric remaindering throughout the algorithm, but only in step 15. The sign adjustments in $E/(m)$ would be a completed wasted until step 15 succeeds. Furthermore, $\chi(\cdot, m)$ can be restricted to the range of $\mathrm{rem}(\cdot, m)$.

Algorithm Modular GCD of section 2 is designed so that it terminates with the proper GCD after all unlucky primes have been eliminated. We comment that there is a practical danger in that if there is a bug in the implementation the algorithm will loop. However, this problem can be diagnosed using an estimate on the number of good primes needed to recover the coefficients of the GCD via condition (6) (for $E = \mathbb{Z}$ multivariate factor coefficient bounds [18, section 4.6.2, exercise 21] can be employed for this purpose) and using an estimate of the number of unlucky primes (see remark following lemma 2 in section 2). In order to keep the code generic, the (possibly optional) estimate computation could be provided by the Euclidean domain.

4.1 Generic Maple Code

The Maple code below is written to be executed with the Domains package in `rmaple`, the current research version of Maple. The Domains package [12] supports the writing of generic code.

The first two inputs A and B are polynomials. The program calculates the $\mathrm{GCD}(A, B)$ where A, B are polynomials in one or more variables $X = x_1, x_2, \ldots, x_n$ over a Euclidean domain E. The third input parameter $P = E[X]$ is the domain in which the polynomials A, B lie. In the program E and X are accessed from the polynomial domain P. The domain X is an ordered Abelian monoid which defines a term ordering on the exponent vectors of the polynomials. The degree function in this setting returns an exponent vector. The operation `V['<'](a,b)` compares exponent vectors a, b.

The fourth input, the domain $G = R[X]$, is the image ring in which the modular GCD computations take place. In the program R is accessed from G. The program assumes that the data structure and term ordering used for $E[X]$ and $G[X]$ are the same, with respect to which the degree vectors and leading coefficients are computed. Our Maple implementation currently does not allow the parameterization of those functions with an admissible ordering. The operation `G['.'](x,a)` means scalar multiplication of a polynomial a by a scalar x.

The fifth input ϕ, the modular homomorphism which maps E into the residue ring $R = E/(m)$, is the remainder function in E. In the Domains package, it must be a canonical simplifier for $E/(m)$.

The sixth input, NextPrime, is a function which must provide an infinite source of unassociated primes in E satisfying (5). The Domains package is designed to permit the modulus m of the residue ring R to be changed by assignment, implicitly redefining the modulus in G, thus without need to construct R and G for each new prime. This is an important design consideration because constructing domains has a cost which may be very high compared with a GCD computation in one variable.

The last input, SymRem, is the symmetric remainder function in E described in (6). For clarity, in our implementation below, this function is passed as a parameter and it is applied to the result of the CRA explicitly.

```
GCD := proc(A,B,P,G,phi,NextPrime,SymRem)

  if A = P[0] then RETURN( P[Normal](B) ) fi;
  if B = P[0] then RETURN( P[Normal](A) ) fi;

  E := P[CoefficientRing]; # Euclidean domain
  R := G[CoefficientRing]; # Residue ring
  X := P[ExponentVector]; # Ordered Abelian monoid

  a := P[Primpart](A,'ca');
  b := P[Primpart](B,'cb');
  gc := E[Gcd](ca,cb); # GCD of contents

  da := P[Degree](a);
  db := P[Degree](b);
  degbound := X[Min](da,db);

  la := P[Lcoeff](a);
  lb := P[Lcoeff](b);
  gamma := E[Gcd](la,lb);

  hbar := P[0]; modulus := E[1];
  for k do

    R[Modulus] := NextPrime();
    while phi(gamma) = E[0] do
      R[Modulus] := NextPrime()
    od;
    m := R[Modulus];

    abar := P[Map](phi,a);
    bbar := P[Map](phi,b);
    userinfo(2,Gcd,"Image GCD computation");
    gbar := G[Gcd](abar,bbar);

    d := G[Degree](gbar); # vector degree
    if d = X[0] then RETURN( P[Constant](gc) ) fi;

    # Leading coefficient correction
    # gbar is assumed to be monic
    gbar := G['.']( phi(gamma), gbar );

    if hbar = P[0] or X['<'](d,degbound) then
       # All previous homomorphism's were unlucky
       degbound := d;
       hbar := P[0];
       modulus := E[1];

    elif X['>'](d,degbound) then
       # This homomorphism is unlucky
       next;
    fi;

    userinfo(2,Gcd,"Chinese remaindering");
    s := R[Inv](phi(modulus));
    v1 := G['.'](s,G['-'](gbar,P[Map](phi,hbar)));
    h := P['+'](hbar,P['.'](modulus,v1));

    modulus := E['*'](m,modulus);
    h := P[Map](proc(x) SymRem(x,modulus) end, h);

    if h = hbar then
      userinfo(2,Gcd,
               "Beginning termination check");
      g := P[Primpart](h);
      if P[Divides](g,a) and P[Divides](g,b) then
# Unit normalization is not multiplicative in Z[i]
        RETURN( P[Normal](P['.'](gc,g)) );
      fi;
    fi;
    hbar := h;
  od;

end:
```

5 Conclusion

We have investigated to which coefficient rings, other than $\mathbb{Z}$, the modular GCD algorithm of Brown can be applied. The main requirement appears to be Euclidean domains which possess a remainder function for which the CRA can reconstruct all values up to a given norm in the Euclidean domain. In particular this includes the Euclidean domains $F[t]$ where F is a field and $\mathbb{Z}[i]$, the Gaussian integers (see [3] for efficient arithmetic in $\mathbb{Z}[i]$ and [16] for efficient arithmetic in any Euclidean quadratic number ring.). The Modular algorithm is not necessarily effective, however, as there may be an intermediate expression swell, e.g., for $F[t] = \mathbb{Q}[t]$.

The Maple code in section 4 gives a clear description of the algorithm which has been tested on univariate and bivariate polynomials for the Euclidean rings $\mathbb{Z}$, $\mathbb{Z}[i]$, $\mathbb{Q}[t]$, $\mathbb{Z}/(p)[t]$. This code applies only one application of the modular GCD algorithm. It is clear that repeated application of it will lead to a method which will not be effective for sparse polynomials. One may combine our generic modular GCD algorithm with the sparse modular GCD algorithm provided any finite fields $E/(p)$ are of sufficient size.

We have shown that when the modular GCD algorithm is applied to the coefficient ring $\mathbb{Z}/(p)[t]$ of $\mathbb{Z}/(p)[t][x]$ when p is too small to to use the modular evaluation algorithm, we obtain a very good algorithm, both in practice and in theory when compared with the approach of computing the GCD in a field extension $\mathrm{GF}(p^k)$.

We note here that we have not used rational reconstruction [23], [16, Section 5], [17, Lemma 1], in the modular GCD algorithm as a means to determine the leading coefficient of the GCD exactly. This is superior because it enables us to reconstruct the GCD with a modulus m of minimal size. Rational reconstruction can be done in Euclidean rings $\mathbb{Z}$, $F[t]$, and $\mathbb{Z}[i]$. It remains to formulate the generic rational reconstruction condition for a corresponding simplifier in an arbitrary Euclidean ring similarly to (6).

The modular algorithm also applies to algebraic number coefficients, which generate rings that may not be unique factorization domains. The GCD is computed over their corresponding algebraic number fields. Several efficient algorithms are described in the literature (see [8, 19] and the papers by Langemyr and McCallum cited there), whose theory generalizes to multivariate polynomials by use of Kronecker's substitution. Our generic algorithm can be adapted to handle such coefficient domains, but we would like to investigate a generic algorithmic theory of algebraic extensions first, so that algebraic function fields are also covered by our implementations.

Acknowledgement Hans Dobbertin sent us a factorization problem of P over $\mathbb{Z}/(2)$ in figure 1, which initiated this work. Barry Trager shared the Axiom code for the modular GCD algorithm with us for comparison. The external and program committee reviewers saw the strengths of this paper through the weaknesses of our presentation, and encouraged us to improve the latter. We thank all of them.

References

Note: many of Erich Kaltofen's publications are accessible through links in the online BibTeX bibliography database at `www.math.ncsu.edu/~kaltofen/bibliography/`.

[1] Bernardin, L., and Monagan, M. B. Efficient multivariate factorization over finite fields. In *Proc. AAECC-12* (Heidelberg, Germany, 1997), vol. 1255 of *Lect. Notes Comput. Sci.*, Springer Verlag, pp. 15–28.

[2] Brown, W. S. On Euclid's algorithm and the computation of polynomial greatest common divisors. *J. ACM 18* (1971), 478–504.

[3] Caviness, B. F., and Collins, G. E. Algorithms for Gaussian integer arithmetic. In *Proc. 1976 ACM Symp. Symbolic Algebraic Comp.* (New York, N. Y., 1976), R. D. Jenks, Ed., ACM, pp. 36–45.

[4] Caviness, B. F., and Rothstein, M. A modular greatest common divisor algorithm for Gaussian polynomials. In *Proc. 1975 ACM Annual Conference* (New York, N. Y., 1975), ACM, pp. 270–273.

[5] Díaz, A., and Kaltofen, E. On computing greatest common divisors with polynomials given by black boxes for their evaluation. In *Proc. 1995 Internat. Symp. Symbolic Algebraic Comput. ISSAC'95* (New York, N. Y., 1995), A. H. M. Levelt, Ed., ACM Press, pp. 232–239.

[6] Díaz, A., and Kaltofen, E. FoxBox a system for manipulating symbolic objects in black box representation. In *ISSAC 98 Proc. 1998 Internat. Symp. Symbolic Algebraic Comput.* (New York, N. Y., 1998), O. Gloor, Ed., ACM Press, pp. 30–37.

[7] Dobbertin, H. Almost perfect nonlinear power functions on $\mathrm{GF}(2^n)$: The Niho case. Manuscript, German Information Security Agency, Bonn, Germany, Apr. 1998.

[8] Encarnación, M. J. Computing GCDs of polynomials over algebraic number fields. *J. Symbolic Comput. 20*, 3 (1995), 299–313.

[9] Gauss, C. F. *Disquisitiones Arithmeticae.* G. Fleischer, Jr., Leipzig, Germany, 1801.

[10] Geddes, K. O., Czapor, S. R., and Labahn, G. *Algorithms for Computer Algebra.* Kluwer Academic Publ., Boston, Massachusetts, USA, 1992.

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

[12] Gruntz, D., and Monagan, M. B. *Introduction to Gauss.* No. 9 in Maple Technical Newsletter. Birkhäuser, 1993, pp. 23–35.

[13] Kaltofen, E. Sparse Hensel lifting. In *EUROCAL 85 European Conf. Comput. Algebra Proc. Vol. 2* (Heidelberg, Germany, 1985), B. F. Caviness, Ed., Lect. Notes Comput. Sci., Springer Verlag, pp. 4–17. Proofs in [14].

[14] Kaltofen, E. Sparse Hensel lifting. Tech. Rep. 85-12, Rensselaer Polytechnic Instit., Dept. Comput. Sci., Troy, N. Y., 1985.

[15] Kaltofen, E. Greatest common divisors of polynomials given by straight-line programs. *J. ACM 35*, 1 (1988), 231–264.

[16] Kaltofen, E., and Rolletschek, H. Computing greatest common divisors and factorizations in quadratic number fields. *Math. Comput. 53*, 188 (1989), 697–720.

[17] Kaltofen, E., and Trager, B. Computing with polynomials given by black boxes for their evaluations: Greatest common divisors, factorization, separation of numerators and denominators. *J. Symbolic Comput. 9*, 3 (1990), 301–320.

[18] Knuth, D. E. *Seminumerical Algorithms*, Third ed., vol. 2 of *The Art of Computer Programming.* Addison Wesley, Reading, Massachusetts, USA, 1997.

[19] Monagan, M., and Margot, R. Computing univariate GCDs over number fields. In *Proc. Symp. Discrete Algo.* (1998), Soc. Indust. Appl. Math, pp. 42–49.

[20] Monagan, M. B. In-place arithmetic for polynomials over $\mathbf{Z}_n$. In *Proc. DISCO '92* (Heidelberg, Germany, 1992), vol. 721 of *Lect. Notes Comput. Sci.*, Springer Verlag, pp. 22–34.

[21] Musser, D. R. *Algorithms for polynomial factorization.* PhD thesis, Univ. Wisconsin, Madison, Wisconsin, USA, 1971. Also TR #134, September 1971.

[22] Rayes, M. O., Wang, P. S., and Weber, K. Parallelization of the sparse modular GCD algorithm for multivariate polynomials on shared memory multiprocessors. In *ISSAC '94 Proc. Internat. Symp. Symbolic Algebraic Comput.* (New York, N. Y., 1994), ACM Press, pp. 66–73.

[23] Wang, P., Guy, M. J. T., and Davenport, J. H. p-adic reconstruction of rational numbers. *SIGSAM Bulletin 16*, 2 (1982).

[24] Zippel, R. Probabilistic algorithms for sparse polynomials. In *Proc. EUROSAM '79* (Heidelberg, Germany, 1979), vol. 72 of *Lect. Notes Comput. Sci.*, Springer Verlag, pp. 216–226.

[25] Zippel, R. Interpolating polynomials from their values. *J. Symbolic Comput. 9*, 3 (1990), 375–403.