J. Symbolic Computation (1989) **8**, 429–448

The Computation of Polynomial Greatest Common Divisors Over an Algebraic Number Field

LARS LANGEMYR *
Numerical Analysis and Computing Science
Royal Institute of Technology
S-100 44 Stockholm, Sweden

SCOTT MCCALLUM †
Institut für Mathematik
Johannes Kepler Universität
A-4040 Linz, Austria

(Received 2 July 1987)

We present a modular algorithm for computing the greatest common divisor of two polynomials over an algebraic number field. Our algorithm is an application of ideas of Brown and Collins. We use the Weinberger-Rothschild homomorphic scheme with the important change that we avoid factoring the modular image of the minimal polynomial. We perform a computing time analysis and report some empirical computing times.

1 Introduction

Algebraic number algorithms tend to be slow. Euclid's algorithm for computing polynomial greatest common divisors (gcd's) over an algebraic number field is no exception. Even for polynomials over the field $\mathbf{Q}$ of rational numbers, Euclid's algorithm can incur explosive coefficient growth. A modular algorithm independently developed by Brown (1971) and Collins (1972) eliminates the problem of coefficient growth in polynomial gcd computation over the rational integers $\mathbf{Z}$. In the present paper we develop a modular algorithm for polynomial gcd's over an algebraic number field using the basic ideas of Brown and Collins.

Let α be a real algebraic number and let $\psi(y)$ be the primitive minimal polynomial of α over $\mathbf{Z}$. Rubald (1974) gave algorithms for arithmetic in the field $\mathbf{Q}(\alpha)$. He also discussed computing gcd's in $\mathbf{Q}(\alpha)[x]$. He described how a generalization of the Brown-Collins method can be used when the following assumptions are satisfied:

*Supported by STU and NSERC. *Present address:* Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Auf der Morgenstelle 10, D-7400 Tübingen.

†Supported by NSERC (Grant 3-640-126-30) and VOEST-ALPINE A.G. (Abt. GCT3). *Formerly with:* Dept. of Computer Science, University of Toronto, Toronto, Canada, M5S 1A4. *Present address:* Research School of Physical Science, Australian National University, Canberra ACT 2601, Australia.

0747–7171/89/110429+20 $03.00/0 © 1989 Academic Press Limited

- $\mathbf{Z}[\theta]$ is a unique factorization domain, where θ is the algebraic integer $\ell\alpha$ (with ℓ the leading coefficient of $\psi(y)$),
- for infinitely many prime numbers p, $\hat{\psi}(y)$ is irreducible modulo p, where $\hat{\psi}(y)$ is the monic irreducible polynomial over $\mathbf{Z}$ satisfied by θ.

Rubald could not find a way to apply the modular method in the cases where the above assumptions are not satisfied. Rubald also gave an algorithm to compute the gcd of two polynomials over an algebraic number field using subresultants. This algorithm uses modular methods to compute a modular representation of the subresultant polynomial remainder sequence (PRS) of the input polynomials considered as bi-variate integral polynomials. From the modular representation of this subresultant PRS the degree of the gcd over $\mathbf{Q}(\alpha)$ is determined, using a partially modular method (a modular algorithm for the gcd of univariate integral polynomials is used). Finally the appropriate subresultant, as determined by the degree, is remaindered with respect to the minimal polynomial using integer arithmetic. Tests done by Rubald indicate that his subresultant algorithm is only better than non-modular methods when the degree of the gcd is relatively small. Rubald gave a bound for the computational complexity of his algorithm. A.K. Lenstra has also studied this problem: he suggests applying the EZGCD algorithm (Loos 1982b). S. Landau has derived a bound on the number of bit operations to compute subresultant remainder sequence for polynomials over algebraic number fields (Landau 1985). In section 5 we compare the bound for the algorithm presented in this paper with the bounds given by Rubald and Landau.

George Collins and Erich Kaltofen suggested to us how to apply the Brown-Collins method to this problem in general. Some theory expounded by Weinberger and Rothschild (1976) holds the key. They derive a bound on the integer denominators of the factors of a given polynomial over $\mathbf{Q}(\alpha)$. Knowing such a bound enables us to dispense with Rubald's first assumption. Weinberger and Rothschild show how use of the Chinese Remainder Theorem makes the second assumption unnecessary also.

Our motivation for seeking the algorithm of the present paper stems from our interest in the cylindrical algebraic decomposition (cad) algorithm (see Collins 1975, and Arnon *et al.* 1984). The cad algorithm makes essential use of algebraic polynomial gcd computation. Thus, in order to extend the range of applicability of the cad algorithm as far as possible, we must use the most efficient gcd algorithm we can find.

We fix some notation that we use throughout the paper. Let $f(x)$ and $g(x)$ be polynomials over a commutative ring R. For $0 \leq j \leq \min(\deg(f), \deg(g))$, we denote by $S_j(f,g)$ the j-th subresultant of f and g, a polynomial of degree at most j whose coefficients are determinants of certain submatrices of the Sylvester matrix of f and g. (see Loos 1982a, Brown and Traub 1971, or Collins 1975, for the exact definition). Let $\mathrm{psc}_j(f,g)$ denote the j-th principal subresultant coefficient of f and g, *i.e.*, the coefficient of x^j in $S_j(f,g)$. Recall that $\mathrm{psc}_0(f,g)$ is just the resultant $\mathrm{res}(f,g)$ of f and g. Let $\mathrm{discr}(f)$ denote the discriminant of f (provided R is an integral domain).

2 Modular Algorithm

Let $\alpha \in \mathbf{C}$ be an algebraic number with minimal polynomial $r(y)$ over $\mathbf{Q}$. Let $f(x)$ and $g(x)$ be non-zero polynomials over $\mathbf{Q}(\alpha)$. The algorithm of this section computes the greatest common divisor of $f(x)$ and $g(x)$.

We replace α by any algebraic integer generating $\mathbf{Q}(\alpha)$. Thus $r(y)$ now has integer coefficients, is monic and irreducible: hence $\mathbf{Z}[\alpha] \simeq \mathbf{Z}[y]/(r(y))$. We take the monic associates of $f(x)$ and $g(x)$ over $\mathbf{Q}(\alpha)$, clear integer denominators, remove integer contents and replace $f(x)$ and $g(x)$ by the results. We can now apply the following algorithm which we assume has access to a list PRIME of distinct odd primes. For certain integers $m \geq 2$ the algorithm is to perform modular arithmetic in the ring $\mathbf{Z}_m$ of integers modulo m. We take as the computational model of $\mathbf{Z}_m$ the set $\{n \in \mathbf{Z} | -m/2 < n \leq m/2\}$.

- Algorithm GCD

- *Inputs:* $r(y)$, a monic irreducible polynomial over $\mathbf{Z}$, $f(x)$ and $g(x)$, non-zero polynomials over $R = \mathbf{Z}[y]/(r(y))$, with integer leading coefficients a and b.

- *Output:* The associate $h(x)$ of the gcd of $f(x)$ and $g(x)$ over the quotient field F of R which has leading coefficient $c = \gcd(a, b)\Delta$, where $\Delta = \mathrm{discr}(r)$ (the coefficients of $h(x)$ lie in R, by Lemma 2.1).

(0) [Initialize.] Set H $\leftarrow$ the empty list. Set P $\leftarrow$ the empty list.

(1) [Compute bound.] Let k be the degree of $h(x)$. Compute (as described in Section 4) a strict bound K for the absolute values of the integer coefficients of $h(x)$, and for the integer R_1, where

$$R_1 = |\mathrm{res}(r(y), \mathrm{psc}_k(f, g))|.^1$$

(2) [Compute modular images.] Let p be the next prime number from the list PRIME.[2] If p divides either a, b or $\mathrm{discr}(r)$, then discard p and begin this step again. Let $\tilde{r}(y) = r(y) \bmod p$. Let $\tilde{R} = \mathbf{Z}_p[y]/(\tilde{r}(y))$ and let $\phi : R \to \tilde{R}$ denote the natural "mod p" homomorphism. (Denote the natural extension of ϕ to $R[x]$ by ϕ also.) Let $\tilde{f} = \phi(f)$ and $\tilde{g} = \phi(g)$. (Note that $\deg(\tilde{f}) = \deg(f)$ and $\deg(\tilde{g}) = \deg(g)$, as p divides neither a nor b, and that $\tilde{r}(y)$ is square-free, as $p \not| \mathrm{discr}(r(y))$.)

(3) [Compute gcd over $\tilde{R}$.] Even though $\tilde{R}$ is not necessarily an integral domain (as $\tilde{r}(y)$ may factor non-trivially over $\mathbf{Z}_p$), one can show that $\tilde{f}(x)$ and $\tilde{g}(x)$ have a gcd $h^*(x)$ over $\tilde{R}$. To do this we use the "Chinese Remainder" Isomorphism

$$\psi : \tilde{R} \to \tilde{R}_1 \times \cdots \times \tilde{R}_t,$$

where $\tilde{r}_1(y), \ldots, \tilde{r}_t(y)$ are the distinct, monic irreducible factors of $\tilde{r}(y)$ over $\mathbf{Z}_p$, and $\tilde{R}_i$ is the field $\mathbf{Z}_p[y]/(\tilde{r}_i(y))$. Denote the natural extension of ψ to $R[x]$ by ψ also. We compute the unique gcd

[1] In this expression $\mathrm{psc}_k(f, g)$ is regarded as an elt. of $\mathbf{Z}[y]$.
[2] If the list PRIME is exhausted then stop and report failure.

$h^*(x)$ of $\tilde{f}(x)$ and $\tilde{g}(x)$ satisfying $\psi_j(h^*) = \gcd(\psi_j(\tilde{f}), \psi_j(\tilde{g}))$, for $1 \leq j \leq t$ (where the ψ_j are the component functions of ψ). We remark that the degree of $h^*(x)$ could be greater than the degree k of $h(x)$ (see Corollary 2.3 and Lemma 2.4 below): in this case, p is called *unlucky*, (and otherwise p is called *lucky*). In practice we might detect an unlucky prime in the course of the computation of $h^*(x)$.

(4) [Compare the degree of h^* with that of its predecessors.] Compare the degree h^* with the degree of the first element h_1^* of H (if there is one). If $\deg(h^*) > \deg(h_1^*)$ then discard $h^*(x)$ and p and return to step 2. If $\deg(h^*) = \deg(h_1^*)$, (or there is no h_1^*) then add h^* to the end of H and add p to the end of P. If $\deg(h^*) < \deg(h_1^*)$ then reset H to the list containing the single element h^* and reset P to the list containing the single element p.

(5) [Test for termination.] Compute the product π of primes in P. If $\pi < 2K$ then return to step 2.

(6) [Chinese Remainder Algorithm.] Let $\mathtt{P} = (p_1, \ldots, p_n)$ and let $\mathtt{H} = (h_1^*, \ldots, h_n^*)$. For $1 \leq i \leq n$ let $c_i = c \bmod p_i$. Let $k^* = \deg(h_1^*)$. Solve the system of congruences.

$$\begin{aligned} h'(x) &\equiv c_1 h_1^*(x) \pmod{p_1} \\ &\vdots \\ h'(x) &\equiv c_n h_n^*(x) \pmod{p_n} \end{aligned}$$

for element $h'(x) \in \mathbf{Z}_\pi[y, x]$ of degree k^* using the Chinese Remainder Algorithm. (In the above system we regard the $h_i^*(x)$ and $h'(x)$ as polynomials in $\mathbf{Z}[y, x]$. Thus the polynomial congruence "$h'(x) \equiv c_i h_i^*(x) \pmod{p_i}$" is really a system of integer congruences, one for each integer coefficient.) Set $h(x) \leftarrow h'(x)$, and exit. □

The termination of the Algorithm GCD is established by the following three lemmas and corollary (the notation of the algorithm is used):

We will first consider the leading coefficient of the gcd $h(x)$. Let $\hat{f}(x)$ and $\hat{g}(x)$ be the monic associates of $f(x)$ and $g(x)$ over $\mathbf{Q}(\alpha)$. Then for $a, b \in \mathbf{Z}$, $\hat{f}(x) \in \frac{1}{a}\mathbf{Z}[\alpha][x]$ and $\hat{g}(x) \in \frac{1}{b}\mathbf{Z}[\alpha][x]$. We establish a lemma on the denominator of the monic gcd $\hat{h}(x)$ over $\mathbf{Q}(\alpha)$ and the cofactors.

Lemma 2.1 *Let $\hat{f}(x) \in \frac{1}{a}\mathbf{Z}[\alpha][x]$ and $\hat{g}(x) \in \frac{1}{b}\mathbf{Z}[\alpha][x]$ be monic. Let $\hat{h}$ be the (monic) gcd of $\hat{f}$ and $\hat{g}$ over $\mathbf{Q}(\alpha)$. Then $\hat{h}(x) \in \frac{1}{c}\mathbf{Z}[\alpha][x]$, where $c = \gcd(a, b)\Delta$. Moreover the cofactors $\check{f} = \hat{f}/\hat{h}$ and $\check{g} = \hat{g}/\hat{h}$ belong to $\frac{1}{a\Delta}\mathbf{Z}[\alpha][x]$ and $\frac{1}{b\Delta}\mathbf{Z}[\alpha][x]$ respectively.*

Proof. This result follows from Lemma 7.1 of Weinberger and Rothschild (1976), together with their observation that the ring of algebraic integers of $\mathbf{Q}(\alpha)$ is contained in $\frac{1}{\Delta}\mathbf{Z}[\alpha]$. □

Lemma 2.2 *Let p be a prime such that p divides neither a, b nor* discr(r). *Then $\phi(h)$ divides h^* in $\tilde{R}[x]$ (with notation as in steps 2 and 3 of the algorithm).*

Proof. Let $\hat{f} = \frac{1}{a}f$, $\hat{g} = \frac{1}{b}g$, and let $\hat{h}$ be the (monic) gcd of $\hat{f}$ and $\hat{g}$ over the quotient field F of R. Let $\check{f} = \hat{f}/\hat{h}$ and $\check{g} = \hat{g}/\hat{h}$. Then by Lemma 2.1, $\hat{h} \in \frac{1}{c}R[x]$, $\check{f} \in \frac{1}{a\Delta}R[x]$ and $\check{g} \in \frac{1}{b\Delta}R[x]$. Hence

$$\begin{array}{rcl} (ca\Delta)\hat{f} & = & (c\hat{h})\ (a\Delta\check{f}) \ \Rightarrow\ c\Delta f = h\bar{f} \\ (cb\Delta)\hat{g} & = & (c\hat{h})\ (b\Delta\bar{g}) \ \Rightarrow\ c\Delta g = h\bar{g}, \end{array}$$

where $\bar{f}$ and $\bar{g}$ are the polynomials $a\Delta\check{f}$ and $b\Delta\check{g}$ (respectively) in $R[x]$. Applying ϕ we obtain

$$\begin{array}{rcl} \phi(c\Delta)\tilde{f} & = & \phi(h)\phi(\tilde{f}) \\ \phi(c\Delta)\tilde{g} & = & \phi(h)\phi(\tilde{g}). \end{array}$$

Thus, as $\phi(c\Delta)$ is a unit of $\tilde{R}$ (as $p \nmid a$, b, discr$(r(y))$), $\phi(h)$ is a common divisor of $\tilde{f}$ and $\tilde{g}$ over $\tilde{R}$. Hence $\phi(h)$ divides h^* in $\tilde{R}[x]$. □

The leading coefficient of $\phi(h)$ is a unit of $\tilde{R}$. Hence we obtain at once:

Corollary 2.3 *With the hypothesis and notation of Lemma 2.2,* $\deg(h^*) \geq k$.

Thus, if p is lucky, then $\deg(h^*) = k$.

Lemma 2.4 *Let p be a prime such that p divides neither a, b nor* $\mathrm{discr}(r)$. *Then* $\deg(h^*) > k$ *if and only if p divides the non-zero integer* $R_1 = \mathrm{res}(r(y), \mathrm{psc}_k(f,g))$ *(with notation as in steps 1, 2 and 3 of GCD).*

Proof. Now $R_1 \neq 0$ because $r(y)$ and $\mathrm{psc}_k(f,g)$ have no common divisor of positive degree in $Z[y]$ (as $r(y)$ is irreducible).

$$\text{Now } p \mid R_1 \iff \phi(\mathrm{psc}_k(f,g)) = 0 \text{ or } (\phi(\mathrm{psc}_k(f,g)) \neq 0 \text{ and } \mathrm{res}(\tilde{r}(y), \phi(\mathrm{psc}_k(f,g))) = 0)\ )^3$$

$$\iff \tilde{r}(y) \text{ and } \phi(\mathrm{psc}_k(f,g)) \text{ have a common divisor in } \mathbf{Z}_p[y] \text{ of positive degree}$$

$$\iff \tilde{r}_j(y) \mid \phi(\mathrm{psc}_k(f,g)) \text{ in } \mathbf{Z}_p[y], \text{ for some } j,\ 1 \leq j \leq t.$$

$$\iff (\psi_j \phi)(\mathrm{psc}_k(f,g)) = 0, \text{ for some } j,\ 1 \leq j \leq t$$

$$\iff \mathrm{psc}_k(\psi_j(\tilde{f}), \psi_j(\tilde{g})) = 0, \text{ for some } j,\ 1 \leq j \leq t$$

$$\iff \deg \gcd(\psi_j(\tilde{f}), \psi_j(\tilde{g})) > k, \text{ for some } j \text{ (as also } \mathrm{psc}_i(\psi_j(\tilde{f}), \psi_j(\tilde{g})) = 0 \text{ for all } i < k)$$

by the Fundamental Theorem of polynomial remainder sequences. (Loos 1982a)

$$\iff \deg(h^*) > k. \quad \square$$

Thus the number of unlucky primes is finite. The following lemma says that when p is lucky, $\phi(c)h^*$ is a bona fide image of the desired gcd h.

Lemma 2.5 *Let p be a prime such that p divides neither a, b nor* $\mathrm{discr}(r)$. *If p is lucky then $\phi(h) = \phi(c)h^*$ (using the notation of steps 2 and 3).*

[3] In this expression $\phi(\mathrm{psc}_k(f,g))$ is regarded as en elt. of $\mathbf{Z}_p[y]$.

Proof. By Lemma 2.2, $h^* = q\phi(h)$, for some $q \in \tilde{R}[x]$. As p does not divide c, the leading coefficient of $\phi(h)$ is $\phi(c)$, a unit of $\tilde{R}$. Hence

$$\deg(h^*) = \deg(q) + \deg(\phi(h)).$$

Hence, as p is lucky, $\deg(q) = 0$. Moreover, as p is lucky, h^* is monic (because $\deg(\gcd(\psi_j(\tilde{f}), \psi_j(\tilde{g}))) = k$, for all j, $1 \leq j \leq t$). Hence $1 = q\phi(c)$, completing the proof. □

We now establish the correctness of Algorithm GCD. In the following we use the notation of step 6. Assume that the algorithm executes step 6. Then

$$\pi = p_1 \cdots p_n \geq 2K.$$

Now $p_1, \ldots, p_n$ are either all lucky or all unlucky. But if $p_1, \ldots, p_n$ are all unlucky, then by Lemma 2.4 π divides the non-zero integer $R_1 = \mathrm{res}(r(y), \mathrm{psc}_k(f,g))$, (as the p_i are distinct). Hence $\pi \leq 2K$, a contradiction. Therefore $p_1, \ldots, p_n$ are all lucky. We must prove that $h'(x) = h(x)$. As K is a strict bound for the absolute values of the integer coefficients of $h(x)$, $h(x) \in \mathbf{Z}_{2K}[y,x]$. Hence, as $2K \leq \pi$, $h(x) \in \mathbf{Z}_\pi[y,x]$. By Lemma 2.5 and the Chinese Remainder Theorem, $h'(x) = h(x)$. The proof of correctness of Algorithm GCD is complete.

3 Refinements to Algorithm

In this section we present two refinements that can be made to Algorithm GCD of section 2. The first refinement is a simplification of the gcd computation over $\tilde{R}$ in step 3. The second is the replacement of the termination test using the a priori bound K by a trial division method.

Some notation first: let $f_1(x)$ and $f_2(x)$ be non-zero polynomials over a field F, with $\deg(f_1) \geq \deg(f_2)$, and let $f_1, f_2, f_3, \ldots, f_{l-1}, f_l$ be the polynomial remainder sequence over F computed by Euclid's algorithm, where the monic associate of f_l is the gcd of f_1 and f_2. Where $n_i = \deg(f_i)$, for $1 \leq i \leq l$, $n_1, n_2, \ldots, n_l$ is called the *degree sequence* of f_1 and f_2 over F. For $0 \leq j \leq n_2$, recall that $S_j(f_1, f_2)$

denotes the j-th subresultant of f_1 and f_2, a polynomial of degree at most j. Let $\mathrm{lsc}_j(f_1, f_2)$ denote the leading (subresultant) coefficient of $S_j(f_1, f_2)$ if $S_j(f_1, f_2) \neq 0$, and zero otherwise. Thus $\mathrm{lsc}_j(f_1, f_2)$ is in general different from the j-th principal subresultant coefficient $\mathrm{psc}_j(f_1, f_2)$ (the coefficient of x^j in $S_j(f_1, f_2)$): $\mathrm{psc}_j(f_1, f_2)$ could vanish, while $\mathrm{lsc}_j(f_1, f_2)$ could be non-zero. By the Fundamental Theorem of PRS's $\mathrm{lsc}_{n_{i-1}-1}(f_1, f_2) \neq 0$ for all i, $3 \leq i \leq l$.

In the following we use the notation of Algorithm GCD. In step 3 we prescribe the computation of the gcd of $\tilde{f}(x)$ and $\tilde{g}(x)$ over $\tilde{R}$ using the Chinese Remainder Isomorphism $\psi : \tilde{R} \rightarrow \tilde{R}_1 \times \cdots \times \tilde{R}_t$. A simpler alternative is to try to compute this gcd within $\tilde{R}[x]$ using Euclid's algorithm. Assume $\deg(f) \geq \deg(g)$. Now the leading coefficient of $\tilde{g}(x)$ (*viz.* $\phi(b)$) is a unit of $\tilde{R}$. Hence one can divide $\tilde{f}(x)$ by $\tilde{g}(x)$ to find $q(x)$ and $s(x)$ in $\tilde{R}[x]$ such that

$$\tilde{f}(x) = \tilde{g}(x)q(x) + s(x),$$

with $\deg(s) < \deg(g)$. Now $s(x)$ may or may not have an invertible leading coefficient. If not, we are stuck, but if so we can divide $\tilde{g}(x)$ by $s(x)$, as in Euclid's algorithm. We will show that, for all but a finite number of primes p it is possible to carry the sequence of divisions in $\tilde{R}[x]$ all the way down to the gcd.

Clearly, it is possible to carry the sequence of division in $\tilde{R}[x]$ all the way down to the gcd exactly when the degree sequence of $\psi_j(\tilde{f})$ and $\psi_j(\tilde{g})$ over $\tilde{R}_j$ is the same for each j, $1 \leq j \leq t$ (that is, independent of j), where ψ_j is the j-th component of ψ, $\psi_j : \tilde{R} \rightarrow \tilde{R}_j$. The following lemma and theorem give a sufficient condition for this to happen.

Lemma 3.1 *Let the degree sequence of $f(x)$ and $g(x)$ over the quotient field F of R be $n_1, n_2, \ldots, n_l = k$, where $n_1 = \deg(f) \geq n_2 = \deg(g)$. Let p be a prime such that $p \not| a$, b, $\mathrm{discr}(r)$. For $3 \leq i \leq l$, let*

$$T_i = \mathrm{res}(r(y), \mathrm{lsc}_{n_{i-1}-1}(f,g)),$$

a non-zero integer. Then

$$p \mid T_i \iff (\psi_j \phi)(\mathrm{lsc}_{n_{i-1}-1}(f,g)) = 0, \text{ for some } j,\ 1 \leq j \leq t.$$

The proof is as in the proof of Lemma 2.4.

Theorem 3.2 *Let the degree sequence of $f(x)$ and $g(x)$ over F be $n_1, n_2, \ldots, n_l = k$, where $n_1 = \deg(f) \geq n_2 = \deg(g)$. Let p be a prime such that p divides neither a, b nor $\mathrm{discr}(r)$. Suppose that p does not divide the non-zero integer T_i defined in the previous lemma, for all i, $3 \leq i \leq l$. Then the degree sequence of $\psi_j(\tilde{f})$ and $\psi_j(\tilde{g})$ over $\tilde{R}_j$ is the same for each j, $1 \leq j \leq t$ (in fact it is $n_1, \ldots, n_l$).*

Proof. Let $1 \leq j \leq t$. Let $r_1, r_2, \ldots, r_m$ be the degree sequence of $\psi_j(\tilde{f})$ and $\psi_j(\tilde{g})$ over $\tilde{R}_j$ Then $n_1 = r_1$ and $n_2 = r_2$ as $p \not| a$, b. Let $3 \leq i \leq \min(l, m)$. Assume that $n_1 = r_1, \ldots, n_{i-1} = r_{i-1}$. Then

$$\begin{aligned}
n_i &= \deg S_{n_{i-1}-1}(f,g) \text{ by the Fundamental Theorem of PRS's.} \\
&= \deg(\psi_j \phi)(S_{n_{i-1}-1}(f,g)) \text{ by Lemma 3.1, as } p \not| T_i \\
&= \deg S_{n_{i-1}-1}(\psi_j(\tilde{f}), \psi_j(\tilde{g})) \text{ as } p \not| a,\ b. \\
&= \deg S_{r_{i-1}-1}(\psi_j(\tilde{f}), \psi_j(\tilde{g})) \text{ as } n_{i-1} = r_{i-1} \\
&= r_i \text{ by the Fundamental Theorem of PRS's.}
\end{aligned}$$

Hence we have $n_i = r_i$ for $1 \leq i \leq \min(l, m)$. Suppose $m < l$. Then

$$\begin{aligned}
(\psi_j \phi)(S_{n_m-1}(f,g)) &= S_{n_m-1}(\psi_j(\tilde{f}), \psi_j(\tilde{g})) \\
&= S_{r_m-1}(\psi_j(\tilde{f}), \psi_j(\tilde{g})) \text{ as } n_m = r_m \\
&= 0 \text{ by the Fundamental Theorem of PRS's.}
\end{aligned}$$

It follows that $(\psi_j\phi)(\mathrm{lsc}_{n_m-1}(f,g)) = 0$. Hence, by Lemma 3.1, $p|T_{m+1}$, contradicting an hypothesis. Suppose $l < m$. then $r_m < r_l = n_l = k$. Therefore $S_{r_m}(f,g) = 0$, by the Fundamental Theorem of PRS's. Hence $S_{r_m}(\psi_j(\tilde{f}), \psi_j(\tilde{g})) = 0$, contradicting the Fundamental Theorem. We conclude that $l = m$.

□

Our simplification to step 3 of GCD consists then in attempting to compute a complete polynomial remainder sequence in $\tilde{R}[x]$ using Euclid's algorithm: should the attempt fail, in case some remainder has a non-invertible leading coefficient, then we must go back to step 2 and try another prime. Theorem 3.2 says that such an attempt will fail for only finitely many primes.

Our second refinement to GCD is to use an alternative termination test. In place of the a priori bound method (step 5) we can use a trial division method, together with an iterative algorithm for the Chinese Remainder Theorem. Instead of retaining the list P of primes processed with minimal degrees so far, and the corresponding list H, we can just retain the product $\bar{p}$ of such primes, and a corresponding polynomial $\bar{h}(x) \in \mathbf{Z}_{\bar{p}}[y,x]$ congruent to $\phi(c)h^*(x)$, for each such prime. After a new prime is processed, yielding a new h^*, if $\deg(h^*) = \deg(\bar{h})$ then the Chinese Remainder Algorithm is applied, yielding say $h'(x) \in \mathbf{Z}_{\bar{p}p}[y,x]$. If $h' = \bar{h}$ then trial divisions over R of $f(x)$ and $g(x)$ by $h'(x)$ are performed. We notice by the proof of Lemma 2.2 that if $h'(x)$ is the correct gcd the quotients $c\Delta f(x)/h'(x)$ and $c\Delta g(x)/h'(x)$ both lie in $R[x]$ so the trial divisions can be performed over the ring R, and not over the quotient field. If $h'(x)$ is a common divisor of $f(x)$ and $g(x)$ then $h(x) = h'(x)$, otherwise another prime is processed. GCD generally processes fewer primes using this trial division method in place of the a priori bound method.

4 Bounds

We will establish bounds that are required to determine the number of primes used by the algorithm. We will also use the bounds to determine an upper bound for the number of unlucky primes. The bounds will thus enable us to analyze the complexity of the refined modular algorithm. We will follow a presentation given by Lenstra (1982).

We will need some definitions: Let α be an algebraic integer and let $r(y) = y^m + \sum_{j=0}^{m-1} \rho_j y^j$, where $\rho_j \in \mathbf{Z}$, be the (monic) minimal polynomial of α. Let m be the degree of $r(y)$. Let the norm $||\beta||$ of an algebraic number field element $\beta \in \mathbf{Q}(\alpha)$, where $\beta = \sum_{i=0}^{m-1} b_i \alpha^i$, be the maximum absolute value of all its conjugates, regarded as elements of C. Thus $||\beta|| = \max_{j=1}^{m} |\sum_{i=0}^{m-1} b_i \alpha_j^i|$, where $b_i \in \mathbf{Q}$, and α_j are the conjugates of α, and $\alpha = \alpha_1$. Consider a polynomial $f'(x) \in \mathbf{Q}(\alpha)[x]$, $f'(x) = \sum_{i=0}^{s'} f_i' x^i$, where $f_i' \in \mathbf{Q}(\alpha)$, and $f_i' = \sum_{j=0}^{m-1} f_{ij}' \alpha^j$, where $f_{ij}' \in \mathbf{Q}$. Define $||f'(x)||_2$, the Euclidean norm of $f'(x)$, to be $\max_{l=1}^{m} (\sum_{i=0}^{s'} |\sum_{j=0}^{m-1} f_{ij} \alpha_l^j|^2)^{1/2}$. This is clearly bounded by $(\sum_{i=0}^{s'} ||f_i'||^2)^{1/2}$. For a polynomial over $\mathbf{Z}$, $g'(y) = \sum_{j=0}^{m'} g_j' y^j$ this norm specializes to $||g'(x)||_2 = (\sum_{j=0}^{m'} |g_j'|^2)^{1/2}$, where $|\cdot|$ is the ordinary absolute value for integers.

We will use the following notation: Use the definitions from the beginning of Section 2. Let $s = \max(\deg(f), \deg(g))$ and let d bound the absolute value of the integer coefficients in both $f(x)$ and $g(x)$ when written as polynomials over $\mathbf{Z}[y]/(r(y))$ as required by the algorithm of Section 2. Let $m = \deg(r)$ as before and let $\hat{d} = \max_{j=0}^{m} |\rho_j|$.

We first give a bound of the absolute value of the integer coefficients in the the algebraic number coefficients of the gcd of $f(x)$ and $g(x)$, $h(x)$. Weinberger and Rothschild (1976) derived such a bound that was later improved by Lenstra (1982) and Landau (1985). The formula is adjusted for the particular leading coefficients obtained from Lemma 2.1. Landau used the fact that Δ in $c = \gcd(a, b)\Delta$ can be

replaced by the largest positive integer δ such that $\delta^2 \mid \Delta$ but we have not used this in our actual implementation of the algorithm.[4]

Proposition 4.1 *We use the notation from above. Then by Lemma 2.1 the gcd of* $f(x)$ *and* $g(x)$ *over* $\mathbf{Q}(\alpha)$ *has an associate* $h(x) \in \mathbf{Z}[\alpha][x]$ *with leading coefficient* $c = \gcd(a,b)\Delta$. *More precisely* $h(x)$ *can be written* $h(x) = \sum_{i=0}^{k} h_i x^i$, *where* $h_i \in \mathbf{Z}[\alpha]$, *and* $h_i = \sum_{j=0}^{m-1} h_{ij}\alpha^j$, *where* $h_{ij} \in \mathbf{Z}$. *Then* $|h_{ij}| \leq K_1$, *where*

$$K_1 = 2^k \Delta^{1/2} \min(||f(x)||_2, ||g(x)||_2) m(m-1)^{(m-1)/2} ||r(y)||_2^{m-1} < 2^{s+3m+1} s d m^{5m+2} \hat{d}^{4m}.$$

Proof. Lenstra proves $|h_{ij}| \leq K_1$ in section 4. We show the inequality to the right. By Mignotte (1982) $||\alpha|| < 1 + \hat{d}$. Hence $\max(||f(x)||_2, ||g(x)||_2) \leq (s+1)dm \max(1, ||\alpha||)^{m-1} < (s+1)dm(1+\hat{d})^{m-1} \leq 2^{m+1} s d m \hat{d}^m$. By the bound of Hadamard (Mignotte 1982) $\mathrm{discr}(r) \leq 2^m m^{3m} \hat{d}^{2m}$. Furthermore $||r(y)||_2 \leq 2m\hat{d}$. To complete the proof we form the product and use $k \leq s$. □

In 2.4 it is shown that p is lucky if it does not divide the integer $R_1 = |\operatorname{res}(r(y), \operatorname{psc}_k(f,g))|$. We now bound R_1.

Proposition 4.2 *Let* $R_1 = |\operatorname{res}(r(y), \operatorname{psc}_k(f,g))|$. *Then* $R_1 < \{2^{m+1} s d m \hat{d}^m\}^{2sm}$.

Proof. Loos (1982b) shows that $\operatorname{res}(r(y), \operatorname{psc}_k(f,g)) = N_\alpha(\operatorname{psc}_k(f,g))$. The norm $N_\alpha(\beta)$ of $\beta \in \mathbf{Q}(\alpha)$ is the product of the conjugates of β. ($N_\alpha(\beta) \in \mathbf{Z}$ if $\beta \in \mathbf{Z}[\alpha]$) Thus $N(\operatorname{psc}_k(f,g)) \leq (||f(x)||_2^s ||g(x)||_2^s)^m$ using Hadamards bound again. As in the proof of Proposition 4.1 $\max(||f(x)||_2, ||g(x)||_2) < 2^{m+1} s d m \hat{d}^m$ and the conclusion follows. □

[4] In fact this can be improved even further by replacing Δ with the defect of α. The defect is however difficult to determine in general.

5 Computing Time Analysis

In this section we analyze the worst case complexity of the algorithm of section 2, with the first improvement only. In the final paragraph of this section we also give an analysis of an algorithm with both refinements, but here we assume that no unlucky event occurs. We use classical arithmetic throughout the section.

For the worst case analysis we will thus not have to consider the cost of the division over $Q(\alpha)$ performed when two consecutive results are equal as described in section 3. Instead we use the termination test $\pi \leq 2K$ from step 5 from section 2. We recommend that a practical implementation uses the second refinement, since experience indicates that we in an average case do not have to use by far as many primes as we would have to if we had used the bound K.

As described in step 2 of the algorithm the selection of primes is made by choosing the primes from a precomputed list. If the list is exhausted, the algorithm fails. It is reasonable to precompute primes that fit into one machine-word on the computer being used. Then the modular arithmetic mod p can be performed in constant time. Let β denote the maximum integer that can be stored in a computer word. We will in the following assume that all p_i are selected from the range $\beta/2 < p_i < \beta$.[5] We will consider the question of how many primes are used by the algorithm. The number of such primes in the above interval is clearly bounded by $\log_{\beta/2}(K)$. Using Proposition 4.1 we obtain that $K_1 < 2^{s+3m+1} s d m^{5m+2} \hat{d}^{4m}$ and by Proposition 4.2 we also bound R_1 by $R_1 < (2^{m+1} s d m \hat{d}^m)^{2ms}$. We have that $\log_{\beta/2}(2^{s+3m+1} s d m^{5m+2} \hat{d}^{4m})$ is $O(s + \log(d) + m\log(m\hat{d}))$. Also $\log_{\beta/2}((2^{m+1} s d m \hat{d}^m)^{2ms})$ is $O(sm\log(ds\hat{d}^m))$. We conclude that the number of primes required to exceed K is $O(sm\log(ds\hat{d}^m))$.

Now consider our first refinement to the algorithm. The number of primes for which the computation can fail is similarly $O(s^2 m \log(ds\hat{d}^m))$ by Lemma 3.1 and Proposition 4.2, since the bound for R_1 bounds

[5] This assumption sets an upper limit on the possible input size of the gcd problem. By using the theory in Rosser & Schoenfeld (1962) we can extend the analysis to be truly asymptotical. This would however not reflect our implementation of the algorithm.

$\mathrm{res}(r(y), \mathrm{lsc}_i(f,g))$ for all i. By the separation of the computation of modular images and application of the Chinese Remainder Algorithm, we only have to lift $O(sm\log(ds\hat{d}^m))$ different modular images however. We are now ready to state the main theorem of this section.

Theorem 5.1 *Consider a version of the algorithm of Section 2, which is implemented using only the first refinement from Section 3. Assume that all primes used by the algorithm are in a fixed range $\beta/2$ and β and that the number of such primes is sufficient for the computation. Then the gcd can be computed in $O(s^4m^3\log^2(ds\hat{d}^m))$ word operations.*

Proof. We consider the complexity of each major step in the algorithm. We compute the image of $f(x)$ and $g(x)$ in $\tilde{R}$ in step 2 The polynomials are of degree $O(s)$ and have maximum integer coefficients of size $O(\log(d))$ and $O(m)$ such integer occurs in each coefficient. Thus one modular image computation requires $O(ms\log(d))$ word operations. The computation of a gcd over $\tilde{R}$ in step 3 costs $O(m^2s^2)$ word operations. In step 6 we lift $h_i(x)$, $1 \leq i \leq n$ to $h(x)$ using the Chinese Remainder Algorithm. This requires $O(ms\log^2(K))$ *i.e.,* $O(s^3m^3\log^2(ds\hat{d}^m))$. We use $O(\log(K))$ primes, lifting integers modulo p_i to integers of size $O(\log(K))$. There are $O(ms)$ such modular numbers to be lifted in the gcd. The number of times a modular gcd computation might fail by hitting zero-divisors is $O(s^2m\log(ds\hat{d}^m))$ so the possible cost of all failing computations is $O(s^4m^3\log(ds\hat{d}^m))$ since one modular gcd costs $O(m^2s^2)$. Thus a total bound for the algorithm is $O(s^4m^3\log^2(ds\hat{d}^m))$. □

We note that this bound is better than the one presented in Landau (1985) for a Subresultant Remainder Sequence over $\mathbf{Q}(\alpha)$. It is also better than the bound presented in Rubald (1974) for his subresultant algorithm. Landau's bound is $O(s^7)$ and Rubald's bound is $O(s^6)$ in the maximum degree of $f(x)$ and $g(x)$.

One empirical observation is that if primes are selected from the range $\beta/2 < p_i < \beta$ *e.g.* where $\beta = 2^{29}$, the primes are almost never unlucky, and also computations almost never fail as described

in Section 3. We apply this observation to obtain a computing time bound which is more similar to the actual computing time. We thus assume that no prime is unlucky and that no failure during computation of the modular remainder sequences occurs. We also assume that both refinements are implemented and that the division test of the second refinement succeeds the first time it is applied, and thus that only two divisions are performed. Then the number of primes used is $O(\log_{\beta/2}(K_1))$ or $O(s + \log(d) + m\log(m\hat{d}))$ since this is the number of primes required to reconstruct the worst case size of the integer coefficients in the gcd. The cost of computing the modular remainder sequences becomes $O(s^2m^2(s + \log(d) + m\log(m\hat{d})))$ and the cost of lifting becomes $O(sm(s + \log(d) + m\log(m\hat{d}))^2)$. For the estimation of the cost of the two divisions we use the assumption that all primes are lucky and thus that the division test will produce an exact quotient. In this case the cost of the division test is of the same order as the cost of computing the product of the exact quotient and the denominator. We note that the integers in the cofactors of the gcd (*i.e.*, the exact quotient) must also be bounded by the factor bound K_1. The cost of multiplying two elements in $\mathbf{Z}[\alpha]$ represented by integers bounded by e in absolute value is $O(m^2(\log(\hat{d})\log(e\hat{d}^m) + \log^2(e)))$. If we insert $\log(e) = O(s + \log(d) + m\log(m\hat{d}))$ the cost of multiplying becomes $O(m^2(s + \log(d) + m\log(m\hat{d})))$. We have to perform $O(s^2)$ such multiplications, and then add $O(s)$ groups of $O(s)$ such results. The addition cost can be ignored however. We now see that the cost of the division test is of the same order as the lifting cost. Thus the total cost in the present case is

$$O(s^2m^2(s + \log(d) + m\log(m\hat{d})) + sm(\log(d) + m\log(m\hat{d}))^2).$$

We claim that this is a resonable estimate of the computing time of an algorithm with both refinements, however certainly no worst case computing time bound.

6 Empirical Computing Times

The refined algorithm of Section 3 has been implemented in ALDES using the SAC-2 library (see Collins 1980). The refined version is clearly easier to implement since we do not have to use the polynomial version of the Chinese Remainder Algorithm to lift the images of $h^*(x)$ in $\tilde{R}_j[x]$, $1 \leq j \leq t$ to $\tilde{R}$, as described in step 3 of the algorithm description of Section 2.

When we use the refined algorithm we compute polynomial remainder sequences in $\tilde{R}[x]$ hoping that we will not hit a leading coefficient which is a zero-divisor. If we hit a zero-divisor we fail and a new prime is selected. We thus need routines for computing in $\tilde{R}$ and in $\tilde{R}[x]$. We have implemented a package that performs these tasks in ALDES. The package handles the case of hitting zero-divisors by returning failure values.

The performance of the refined version of the algorithm has been tested on a large number of randomly generated polynomials. Its performance has been compared to the algorithm originally available with the SAC-2 library. That algorithm uses a monic Euclidean polynomial remainder sequence in $\mathbf{Q}(\alpha)[x]$. The performance obtained is listed in the table below. The tests have been done in the spirit of Rubald (1974), *i.e.*, in the following way: We generate three random polynomials h, $\tilde{f}$ and $\tilde{g}$ over $\mathbf{Z}[\alpha]$ with h of degree $\mathcal{N}$ and $\tilde{f}$ and $\tilde{g}$ of degree $\mathcal{M}$. All three polynomials have integers with $\mathcal{K}$ decimal digits in the coefficients. We then form the input to the gcd algorithm: $f = h\tilde{f}$ and $g = h\tilde{g}$.We are thus able to control the degree of the gcd h with high probability. After the gcd has been computed we can easily check that $\tilde{f}$ and $\tilde{g}$ have no factors in common. In the table below the timings for both the modular and non-modular algorithm is shown in seconds. The timings were obtained by calculating the mean of 10 consecutive runs with the same input sizes, but with different randomly generated integers in the coefficients. The rows contain timings for $\mathcal{N} = 0, 2, 4, 6$ and the columns contain timings for $\mathcal{M} = 2, 4, 6, 8, 10$. Each row has three sub-rows where $\mathcal{K} = 5, 10, 15$. The figure on the left side of each column is the time for the

modular algorithm, and to the right there is the timing for the non-modular algorithm. We note that the performance of the modular algorithm is significantly better than that of the non-modular algorithm. The algebraic number field in which the calculations were done is $\mathbf{Q}(i)$ *i.e.*, $r(y) = y^2 + 1$.

$\mathcal{N}$	$\mathcal{K}\backslash\mathcal{M}$	2		4		6		8		10	
0	5	0.3	0.6	0.6	3.7	1.2	15	1.6	39	1.9	93
	10	0.6	1.3	0.9	10	1.4	43	2.2	120	2.7	250
	15	0.8	2.0	1.2	17	2.2	73	2.9	210	3.5	500
2	5	1.5	1.4	2.8	8.2	4.0	27	6.1	71	8.1	160
	10	2.4	3.8	3.7	24	5.7	87	7.6	200	9.7	440
	15	2.9	6.8	4.5	43	6.9	150	8.9	410	12	890
4	5	2.3	2.6	4.0	14	6.4	43	8.6	109	10	220
	10	3.5	5.8	5.8	35	8.1	122	11	290	13	329
	15	4.6	10	7.1	62	10	217	14	540	16	1100
6	5	3.3	3.1	5.7	17	8.0	56	11	123	13	220
	10	4.7	8.6	7.5	46	10	150	14	363	17	747
	15	6.5	14	10	87	13	280	17	721	22	1400

The timings were obtained on a `sun3/75` workstation, running version 3.0 of the `sun OS`, using the `f77` compiler without optimization. We also tested the algorithm with an extension of higher degree. We let $\alpha \in \mathbf{C}$ be a root of the polynomial $r(y) = y^5 - y + 1$ over $\mathbf{Q}$. The test was performed using the same strategy as in the previous example but only with $\mathcal{N} = 0, 2$, $\mathcal{M} = 2, 4, 6, 8$ and $\mathcal{K} = 5, 10$.

$\mathcal{N}$	$\mathcal{K}\backslash\mathcal{M}$	2		4		6		8	
0	5	5.8	22	8.9	180	13	790	16	2300
	10	13	55	19	530	24	2200	32	–
2	5	14	48	23	370	31	1300	43	3700
	10	30	140	46	1000	64	4000	84	–

Additional experiments indicate that the performance advantage is even better for the modular algorithm when $r(y)$ has higher degree or larger coefficient size.

Some tests has been made to determine the improvement on the cylindrical algebraic decomposition (cad) algorithm (Collins 1975, and Arnon *et al.* 1984) using our modular gcd algorithm. In the cad method the gcd algorithm is required to make polynomials over algebraic number fields square-free. A few example runs have been tested using the SAC-2 implementation of the cad algorithm. The tests have not been done as systematically as above. They however indicate that we get a substantial performance increase when the modular algorithm is used. It was pointed out by McCallum (1985) that the calculations of gcd's of polynomials over algebraic number fields often require a major part of the

execution time of the whole cad algorithm, thus an improvement of the gcd algorithm would improve the performance of the whole algorithm significantly. This observation has been confirmed by our tests.

We are grateful to George Collins and Erich Kaltofen for pointing to the work of Weinberger and Rothschild. George Collins also corrected our description of the Rubald algorithm, and helpfully provided advice for the implementation. Thanks to Joachim von zur Gathen for suggesting the first refinement to our algorithm presented in Section 3. Lars Langemyr is grateful to Prof. Derek Corneil for inviting him to University of Toronto the spring 1986 and to the computer algebra group at University of Bath for discussing a draft of the manuscript.

Bibliography

Arnon, D. S., Collins, G. E., McCallum, S., 1984. Cylindrical algebraic decomposition. *SIAM J. on Comp.*, **13**, 865–877, 878–889.

Brown, W. S, 1971. On Euclid's algorithm and the computation of polynomial greatest common divisors. *J. of the ACM*, **18**/4, 478–504.

Brown, W. S., Traub, F. J, 1971. On Euclid's algorithm and the theory of subresultants. *J. of the ACM*, **18**/4, 505–514.

Collins, G. E., 1972. *The SAC–1 Polynomial GCD and Resultant System.* Technical Report University of Wisconsin, MACC Tech. Report 27, University of Wisconsin, Madison.

Collins, G. E., 1975. Quantifier elimination for real closed fields by cylindrical algebraic decomposition. In *Second GI Conf. Automata Theory and Formal Languages*, 134–183, Springer-Verlag, Lecture Notes in Computer Science 33.

Collins, G. E., 1980. ALDES and SAC–2 now available. *SIGSAM bull.*, **12**/2, 19.

Landau, S., 1985. Factoring polynomials over algebraic number fields. *SIAM J. on Comp.*, **14**, 184–195.

Lenstra, A. K., 1983. Factoring polynomials over algebraic number fields. In *Proc. EUROCAL '83*, 245–254, Springer-Verlag, Lecture Notes in Computer Science 162.

Loos, R. G. K., 1982a. Generalized polynomial remainder sequences. In B. Buchberger, G. E. Collins, and R. G. K. Loos, editors, *Computer Algebra, Symbolic and Algebraic Computation*, 115–137, Springer-Verlag, Wien-New York.

Loos, R. G. K., 1982b. Computing in algebraic extensions. In B. Buchberger, G. E. Collins, and R. G. K. Loos, editors, *Computer Algebra, Symbolic and Algebraic Computation*, 173–187, Springer-Verlag, Wien-New York.

McCallum, S., 1985. *An Improved Projection Operation for Cylindrical Algebraic Decomposition.* PhD thesis, University of Wisconsin, Madison.

Mignotte, M, 1982. Some useful bounds. In B. Buchberger, G. E. Collins, and R. G. K. Loos, editors, *Computer Algebra, Symbolic and Algebraic Computation*, 259–263, Springer-Verlag, Wien-New York.

Rosser, J. B., Schoenfeld, L., 1962. Approximate formulas for some functions of prime numbers. *Illinois J. Math.*, 6, 64–94.

Rubald, C. M, 1974. *Algorithms for Polynomials Over a Real Algebraic Number Field.* PhD thesis, University of Wisconsin, Madison.

Weinberger, P. J., Rothschild, L. P., 1976. Factoring polynomials over algebraic number fields. *ACM Transactions on Mathematical Software*, 2/4, 335–350.