On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors

W. S. BROWN

Bell Telephone Laboratories, Incorporated, Murray Hill, New Jersey

ABSTRACT. This paper examines the computation of polynomial greatest common divisors by various generalizations of Euclid's algorithm. The phenomenon of coefficient growth is described, and the history of successful efforts first to control it and then to eliminate it is related.

The recently developed modular algorithm is presented in careful detail, with special attention to the case of multivariate polynomials.

The computing times for the classical algorithm and for the modular algorithm are analyzed, and it is shown that the modular algorithm is markedly superior. In fact, in the multivariate case, the maximum computing time for the modular algorithm is strictly dominated by the maximum computing time for the first pseudo-division in the classical algorithm.

KEY WORDS AND PHRASES: algebra, asymptotic bounds, Chinese remainder algorithm, coefficient growth, computing time analysis, Euclid's algorithm, greatest common divisors, intermediate expression swell, modular arithmetic, modular mappings, polynomial remainder sequences, polynomials, subresultants

CR CATEGORIES: 5.0, 5.9

1. *Introduction*

1.1 ORIGIN AND SCOPE. According to Knuth [1, p. 294], "Euclid's algorithm, which is found in Book 7, Propositions 1 and 2 of his *Elements* (c. 300 B.C.), and which many scholars conjecture was actually Euclid's rendition of an algorithm due to Eudoxus (c. 375 B.C.), . . . is the oldest nontrivial algorithm which has survived to the present day."

The algorithm, as presented by Euclid, computes the positive greatest common divisor of two given positive integers. However, it is readily generalized to apply to polynomials in one variable over a field, and further to polynomials in any number of variables over any unique factorization domain in which greatest common divisors can be computed.

In particular, we shall focus our attention on domains of *univariate* or *multivariate* polynomials (that is, polynomials in one or several variables, respectively) over the integers or the rationals, even though some of the results will be stated more generally.

We shall not consider polynomials with real coefficients, because the reals are

This paper is a revised version of one with the same title that appears in "The Proceedings of the Second Symposium on Symbolic and Algebraic Manipulation, March 23–25, 1971," held by ACM-SIGSAM, which was prepared before the symposium.

not exactly representable in a computer, and any use of finite precision approximations can make it impossible to test whether one polynomial divides another, or even to determine the degree of a polynomial. Although there may be special situations in which these obstacles can usefully be attacked, it is clear that they cannot be overcome in any general way.

1.2 APPLICATIONS. The problem of computing GCD's has recently received considerable attention because of the need to simplify rational numbers (quotients of integers) and rational functions (quotients of polynomials) in systems for mechanized algebra [2, 12]. However, Euclid's algorithm is also intimately connected with continued fractions [1, pp. 316–320], Diophantine equations [1, p. 303], bigradients [3, 4], Sturm sequences [5, Ch. 7], and elimination theory [5, Ch. 12] (including resultants and discriminants).

1.3 BASIC CONCEPTS. Before proceeding further, let us review some basic concepts [6, Ch. 3]. In an integral domain, divisors of unity are called *units*, and elements which divide each other are called *associates*; clearly, the ratio of two associates is a unit. In a field, all nonzero elements are units; in the domain of integers, however, the only units are 1 and -1.

An element of an integral domain is said to be *irreducible* if its only divisors are units and associates. An integral domain in which every element can be represented uniquely (up to associativity) as a product of irreducibles is called a *unique factorization domain.*

The relation of associativity is an equivalence relation, and can therefore be used to decompose an integral domain into *associate classes.* It is often convenient to single out one element of each associate class as a canonical representative, and define it to be *unit normal.* In the domain of integers, all nonnegative integers are unit normal. In a field, since all nonzero elements are units, only 0 and 1 are unit normal. In a domain of polynomials, those with unit-normal leading coefficients are unit normal.

Let $a_1, \cdots, a_n$ be given nonzero elements of an integral domain. Then g is called the *greatest common divisor* (GCD) of $a_1, \cdots, a_n$ if and only if

(a) g divides $a_1, \cdots, a_n$,
(b) every divisor of $a_1, \cdots, a_n$ divides g, and
(c) g is unit normal.

In a unique factorization domain, there is always a unique GCD $g = \gcd(a_1, \cdots, a_n)$. If $g = 1$, we say that $a_1, \cdots, a_n$ are *relatively prime.*

1.4 THE ALGORITHM FOR INTEGERS. Let a_1 and a_2 be positive integers with $a_1 \geq a_2$, and let $\gcd(a_1, a_2)$ denote their (positive) greatest common divisor. To compute this GCD, *Euclid's algorithm* constructs the *integer remainder sequence* (IRS) $a_1, a_2, \cdots, a_k$, where a_i is the positive remainder from the division of a_{i-2} by a_{i-1}, for $i = 3, \cdots, k$, and where a_k divides a_{k-1} exactly. That is,

$$a_i = a_{i-2} - q_i a_{i-1}, \qquad 0 < a_i < a_{i-1}, \quad i = 3, \cdots, k, \tag{1}$$

and $a_k \mid a_{k-1}$. From this it is easy to see that $\gcd(a_1, a_2) = \gcd(a_2, a_3) = \cdots = \gcd(a_{k-1}, a_k) = a_k$, and therefore a_k is the desired GCD.

This algorithm can be extended [1, p. 302] to yield integers u_i and v_i such that

$$u_i a_1 + v_i a_2 = a_i, \qquad i = 1, \cdots, k. \tag{2}$$

When $\gcd(a_1, a_2) = 1$, it follows that $u_k a_1 + v_k a_2 = 1$, and therefore u_k is an inverse of a_1 modulo a_2, while v_k is an inverse of a_2 modulo a_1. If only u_k is needed, as in Step (1) of the Chinese remainder algorithm (Section 4.8), then one need not compute $v_1, \cdots, v_k$; if $a_1 \gg a_2$, the time saved may be substantial.

1.5 THE ALGORITHM FOR POLYNOMIALS. We shall consider two fundamentally different generalizations of Euclid's algorithm (Section 1.4) to domains of polynomials.

In the classical algorithm (Section 2), we view a multivariate polynomial as a univariate polynomial with polynomial coefficients, and we construct a sequence of polynomials of successively smaller degree. Unfortunately, as the polynomials decrease in degree, their coefficients (which may themselves be polynomials) tend to grow, so the successive steps tend to become harder as the calculation progresses.

If the GCD's of these inflated coefficients are required, the problem is aggravated—especially in the multivariate case, where the growth may be compounded through several levels of recursion. If the coefficient domain is a field, this same remark applies to any GCD's of numerators and denominators that are required to simplify inflated coefficients. If coefficients in a field are not simplified, then the division steps become harder faster, and the final result, although formally correct, may be practically useless.

In the modular algorithm (Section 4) we first project the given polynomials into one or more simpler domains in which images of the GCD can more easily be computed. The true GCD is then constructed from these images with the aid of the Chinese remainder algorithm. Since the same method is used for the required GCD computations in the image spaces, it is only necessary to apply Euclid's algorithm to integers and to univariate polynomials with coefficients in a finite field.

1.6 RECENT HISTORY. During the past decade these algorithms have been studied intensively by G. E. Collins, and (mostly in response to Collins' work) by the author.

The first major advance was the discovery by Collins [7] of the subresultant PRS algorithm (Section 3.6), which effectively controls coefficient growth without any GCD computations in the coefficient domain or any subdomain thereof. Then, after several years of improvement and consolidation, Collins and the author (working independently but with some communication) discovered the essentials of the modular algorithm (Section 4) which completely eliminates the problem of coefficient growth by using modular arithmetic. With a very few hints from Collins and the author, D. E. Knuth immediately grasped most of the key ideas and published a sketch of a similar algorithm [1, pp. 393–395].

1.7 OUTLINE. In Section 2, we present the classical algorithm without specifying a method for constructing the required sequence of polynomials. Several algorithms for this purpose are discussed in Section 3, culminating in the subresultant PRS algorithm mentioned above. In Section 4, the modular algorithm is presented in careful detail, with special attention to the multivariate case. The required computing times for the classical algorithm (augmented by the subresultant PRS algorithm) and for the modular algorithm are then analyzed in Section 5. Finally, in Section 6 we review the highlights and present some tentative conclusions.

2. *The Classical Algorithm*

2.1 INTRODUCTION. The goal of this section is to obtain a straightforward generalization of Euclid's algorithm, as presented in Section 1.4, to domains of univariate or multivariate polynomials. By viewing multivariate polynomials as polynomials in one variable, hereafter called the *main variable*, with coefficients in the domain of polynomials in the other variables, hereafter called the *auxiliary variables*, we may confine our attention without loss of generality to the univariate case.

2.2 THE RATIONAL ALGORITHM. For univariate polynomials F_1 and F_2 over a field, division yields a unique quotient Q and remainder R such that

$$F_1 = QF_2 + R, \qquad \partial(R) < \partial(F_2), \tag{3}$$

where $\partial(F)$ denotes the degree of F, and $\partial(0) = -\infty$. Thus Euclid's algorithm, as presented in Section 1.4, is directly applicable. As an example [1, pp. 370–371], if

$$\begin{aligned} F_1(x) &= x^8 + x^6 - 3x^4 - 3x^3 + 8x^2 + 2x - 5, \\ F_2(x) &= 3x^6 + 5x^4 - 4x^2 - 9x + 21 \end{aligned} \tag{4}$$

are viewed as polynomials with rational coefficients, then the following sequence occurs (for brevity, we write only the coefficients):

$$\begin{array}{l} 1, 0, 1, 0, -3, -3, 8, 2, -5 \\ 3, 0, 5, 0, -4, -9, 21 \\ -\frac{5}{9}, 0, \frac{1}{9}, 0, -\frac{1}{3} \\ -\frac{117}{25}, -9, \frac{441}{25} \\ \frac{233150}{6591}, -\frac{102500}{2197} \\ \frac{1288744821}{543589225}. \end{array} \tag{5}$$

It follows that F_1 and F_2 are relatively prime.

To improve this procedure, we make each polynomial monic as soon as it is obtained, thereby simplifying the coefficients somewhat. In our example, we obtain the sequence

$$\begin{array}{l} 1, 0, 1, 0, -3, -3, 8, 2, -5 \\ 1, 0, \frac{5}{3}, 0, -\frac{4}{3}, -3, 7 \\ 1, 0, -\frac{1}{5}, 0, \frac{3}{5} \\ 1, \frac{25}{13}, -\frac{49}{13} \\ 1, -\frac{6150}{4663} \\ 1. \end{array} \tag{6}$$

Although this sequence minimizes the growth of coefficients, it requires integer GCD computations at each step in order to reduce the fractions to lowest terms.

In general, if the coefficient domain is not a field, we could embed it in its field of quotients and then use the algorithm of Section 1.4, but we shall see that it is more efficient not to do so.

2.3 POLYNOMIAL REMAINDER SEQUENCES. Let $\mathcal{I}$ be a unique factorization domain in which there is some way of finding GCD's, and let $\mathcal{I}[x]$ denote the domain of polynomials in x with coefficients in $\mathcal{I}$. Assuming that the terms of a polynomial $F \in \mathcal{I}[x]$ are arranged in the order of decreasing exponents, the first term is called the *leading term*, and its coefficient $\mathrm{lc}(F)$ is called the *leading coefficient*.

Since the familiar process of polynomial division with remainder requires exact divisibility in the coefficient domain, it is usually impossible to carry it out for nonzero $F_1, F_2 \in \mathcal{I}[x]$. However, the process of *pseudo-division* [1, p. 369] always yields a unique *pseudo-quotient* $Q = \mathrm{pquo}(F_1, F_2)$ and *pseudo-remainder* $R = \mathrm{prem}\,(F_1, F_2)$, such that $f_2^{\delta+1}F_1 - QF_2 = R$ and $\partial(R) < \partial(F_2)$, where f_2 is the leading coefficient of F_2, and $\delta = \partial(F_1) - \partial(F_2)$.

For nonzero $F_1, F_2 \in \mathcal{I}[x]$, we say that F_1 is *similar* to F_2 ($F_1 \sim F_2$) if there exist $a_1, a_2 \in \mathcal{I}$ such that $a_1F_1 = a_2F_2$. Here a_1 and a_2 are called *coefficients of similarity.*

For nonzero $F_1, F_2 \in \mathcal{I}[x]$ with $\partial(F_1) \geq \partial(F_2)$, let $F_1, F_2, \cdots, F_k$ be a sequence of nonzero polynomials such that $F_i \sim \mathrm{prem}(F_{i-2}, F_{i-1})$ for $i = 3, \cdots, k$, and $\mathrm{prem}(F_{k-1}, F_k) = 0$. Such a sequence is called a *polynomial remainder sequence* (PRS). From the definitions, it follows that there exist nonzero $\alpha_i, \beta_i \in \mathcal{I}$ and $Q_i \sim \mathrm{pquo}(F_1, F_2)$ such that

$$\beta_i F_i = \alpha_i F_{i-2} - Q_i F_{i-1}, \qquad \partial(F_i) < \partial(F_{i-1}), \quad i = 3, \cdots, k. \tag{7}$$

Because of the uniqueness of pseudo-division, the PRS beginning with F_1 and F_2 is unique up to similarity. Furthermore, it is easy to see that $\gcd(F_1, F_2) \sim \gcd(F_2, F_3) \sim \cdots \sim \gcd(F_{k-1}, F_k) \sim F_k$. Thus, the construction of the PRS yields the desired GCD to within similarity.

2.4 ALGORITHM C. A polynomial $F \in \mathcal{I}[x]$ will be called *primitive* if its nonzero coefficients are relatively prime; in particular, all polynomials over a field are primitive. Since $\mathcal{I}$ is a unique factorization domain, it follows [6, pp, 74–77] that $\mathcal{I}[x]$ is a unique factorization domain whose units are the units of $\mathcal{I}$. Hence each polynomial $F \in \mathcal{I}[x]$ has a unique representation of the form $F = \mathrm{cont}(F)\mathrm{pp}(F)$, where $\mathrm{cont}(F)$ is the (unit normal) GCD of the coefficients of F, and $\mathrm{pp}(F)$ is a primitive polynomial. We shall refer to $\mathrm{cont}(F)$ and $\mathrm{pp}(F)$ as the *content* and *primitive part,* respectively, of F.

Let F_1' and F_2' be given nonzero polynomials in $\mathcal{I}[x]$ with $\partial(F_1') \geq \partial(F_2')$, and let G' be their GCD. Also, let $c_1 = \mathrm{cont}(F_1')$, $c_2 = \mathrm{cont}(F'_2)$, $c = \gcd(c_1, c_2)$, $F_1 = \mathrm{pp}(F_1')$, $F_2 = \mathrm{pp}(F_2')$, and $G = \gcd(F_1, F_2)$. Now, if $F_1, F_2, \cdots, F_k$ is a PRS, it is easy to show that $G = \mathrm{pp}(F_k)$ and $G' = cG$. Because of coefficient growth, the coefficients of F_k are likely to be much larger than those of F_1 and F_2. However, since G divides both F_1 and F_2, the coefficients of G are usually smaller than those of F_1 and F_2. Thus, F_k is likely to have a very large content. Fortunately, most of this unwanted content can be removed without computing any GCD's involving coefficients of F_k.

Let $f_1 = \mathrm{lc}(F_1)$, $f_2 = \mathrm{lc}(F_2)$, $f_k = \mathrm{lc}(F_k)$, $g = \mathrm{lc}(G)$, and $\bar{g} = \gcd(f_1, f_2)$. Since G divides both F_1 and F_2, it follows that g divides both f_1 and f_2, and therefore $g \mid \bar{g}$. Let $\bar{G} = (\bar{g}/g)G$. Clearly, $\bar{G}$ has $\bar{g}$ as its leading coefficient, and G as its primitive part, and it is easy to see that $\bar{G} = \bar{g}F_k/f_k$.

In the case of the Euclidean PRS algorithm (Section 3.2), the reduced PRS algorithm (Section 3.4), and the subresultant PRS algorithm (Section 3.6), it can be shown (see [7] and [8]) that $\bar{g}$ divides a subresultant (Section 3.5) which in turn divides F_k, and therefore $\bar{g} \mid f_k$. Hence, $\bar{G} = F_k/(f_k/\bar{g})$. Thus, the (large) factor $f_k/\bar{g}$ can be removed for the price of computing $\bar{g}$ and performing some divisions, and it remains only to remove the (relatively small) content of $\bar{G}$.

In other cases, such as the primitive PRS algorithm (Section 3.3), if $\bar{g}$ does not divide f_k, we may compute $\bar{G}$ directly from the formula $\bar{G} = \bar{g}F_k/f_k$. Alternatively, dividing the numerator and denominator of this expression by $h = \gcd(f_k, \bar{g})$, we obtain $\tilde{G} = ((\bar{g}/h)F_k)/(f_k/h)$. Since (f_k/h) and $\bar{g}/h)$ are relatively prime, it follows that (f_k/h) divides F_k. Hence, we may obtain G by computing $H = F_k/(f_k/h)$, and then $G = \mathrm{pp}(H)$.

We are now prepared to present Algorithm C for computing $G' = \gcd(F_1', F_2')$:

(1) Set $c_1 = \mathrm{cont}(F_1')$, $c_2 = \mathrm{cont}(F_2')$, $c = \gcd(c_1, c_2)$.
(2) Set $F_1 = F_1'/c_1$, $F_2 = F_2'/c_2$.
(3) Set $f_1 = \mathrm{lc}(F_1)$, $f_2 = \mathrm{lc}(F_2)$, $\bar{g} = \gcd(f_1, f_2)$.
(4) Construct a PRS, $F_1, F_2, \cdots, F_k$.
(5) If $\partial(F_k) = 0$, set $G' = c$, and return.
(6) Set $\tilde{G} = F_k/(f_k/\bar{g})$ or $\bar{g}F_k/f_k$ as appropriate.
(7) Set $G = \mathrm{pp}(\tilde{G}) = \tilde{G}/\mathrm{cont}(\tilde{G})$.
(8) Set $G' = cG$, and return.

3. *Constructing a PRS*

3.1 INTRODUCTION. We turn now to the problem of constructing a PRS, as required in Step 4 of Algorithm C.

Let $F_1, F_2, \cdots, F_k$ be a PRS in $\mathcal{g}[x]$. Let $d_i = \partial(F_i)$, for $i = 1, \cdots, k$, and note that $d_1 \geq d_2 > d_3 > \cdots > d_k \geq 0$. Let $\delta_i = d_i - d_{i+1}$ for $i = 1, \cdots, k-1$, and note that $\delta_1 \geq 0$, while $\delta_i > 0$ for $i > 1$. If $\delta_i = 1$ for all $i > 1$, the PRS is called *normal*; otherwise it is called *abnormal*. Finally, let f_i denote the leading coefficient of F_i.

Referring to (7), we shall always choose

$$\alpha_i = f_{i-1}^{\delta_{i-2}+1}, \qquad i = 3, \cdots, k, \tag{8}$$

so that

$$\beta_i F_i = \mathrm{prem}(F_{i-2}, F_{i-1}), \qquad i = 3, \cdots, k. \tag{9}$$

When a method for choosing the β_i is given, this equation and the terminating condition

$$\mathrm{prem}(F_{k-1}, F_k) = 0 \tag{10}$$

together provide an algorithm for constructing the PRS. We shall consider several different methods for choosing the β_i.

3.2 THE EUCLIDEAN PRS ALGORITHM. If we choose $\beta_i = 1$ for $i = 3, \cdots, k$, then $F_i = \mathrm{prem}(F_{i-2}, F_{i-1})$ for $i = 3, \cdots, k$. Collins calls this the *Euclidean PRS algorithm* because it is the most obvious generalization of Euclid's algorithm to polynomials over a unique factorization domain that is not a field.

Returning to the example (4), the Euclidean PRS is

$$\begin{array}{l} 1, 0, 1, 0, -3, -3, 8, 2, -5 \\ 3, 0, 5, 0, -4, -9, 21 \\ -15, 0, 3, 0, -9 \\ 15795, 30375, -59535 \\ 1254542875143750, -1654608338437500 \\ 12593338795500743100931141992187500. \end{array} \tag{11}$$

Although the Euclidean PRS algorithm is easy to state, it is thoroughly impractical since the coefficients grow exponentially as we proceed through the sequence. The only comparably bad method with any surface plausibility would be to work over the field of quotients of $\mathcal{I}$, as in (5), but without simplification.

3.3 THE PRIMITIVE PRS ALGORITHM. To obtain a PRS with minimal coefficient growth, we choose $\beta_i = \operatorname{cont}(\operatorname{prem}(F_{i-2}, F_{i-1}))$ for $i = 3, \cdots, k$, so that $F_3, \cdots, F_k$ are primitive. This is called the *primitive PRS algorithm*. In the example (4), it yields the sequence

$$\begin{array}{l} 1, 0, 1, 0, -3, -3, 8, 2, -5 \\ 3, 0, 5, 0, -4, -9, 21 \\ 5, 0, -1, 0, 3 \\ 13, 25, -49 \\ 4663, -6150 \\ 1. \end{array} \tag{12}$$

Unfortunately, it is necessary to calculate one or more GCD's of coefficients at each step, and these become progressively harder as the coefficients grow. However, since the growth is essentially linear (Section 3.5), these GCD's would be well worth the effort if the Euclidean PRS algorithm were the only alternative.

In the multivariate case, the linear coefficient growth is, of course, compounded through each level of the recursion, but even so, the primitive PRS algorithm has demonstrated considerable practical utility.

3.4 THE REDUCED PRS ALGORITHM. Since the bulk of the work in the primitive PRS algorithm is in primitive part computations, we would like to find a way to avoid most of them and still reduce the coefficient growth sharply from that which occurs in the Euclidean PRS algorithm.

Surprisingly, we can accomplish this to a significant extent by choosing

$$\begin{aligned} \beta_3 &= 1, \\ \beta_i &= \alpha_{i-1}, \qquad i = 4, \cdots, k. \end{aligned} \tag{13}$$

This is Collins' *reduced PRS algorithm*, and is justified in [7] and [8] by a proof that is sketched in Section 3.5.

In a normal reduced PRS, the coefficient growth is essentially linear (see Section 3.5). In an abnormal reduced PRS, the growth can be exponential, but of course not as badly so as in the corresponding Euclidean PRS. In the example (4), which is distinctly abnormal, the reduced PRS is

$$\begin{array}{l} 1, 0, 1, 0, -3, -3, 8, 2, -5 \\ 3, 0, 5, 0, -4, -9, 21 \\ -15, 0, 3, 0, -9 \\ 585, 1125, -2205 \\ -18885150, 24907500 \\ 527933700. \end{array} \tag{14}$$

Notice that the coefficient sizes appear to be doubling at each step until the last. It is easy to show that the small size of F_6 results from the fact that $\delta_4 < \delta_3$. If all of the δ_i were the same and greater than 1, then the growth would be uniformly exponential.

3.5 SUBRESULTANTS. Let $F_1, F_2, \cdots, F_k$ be any PRS in $\mathcal{I}[x]$. The major result of Collins [7] is that

$$F_i \sim S_{d_i}(F_1, F_2) \sim S_{d_{i-1}-1}(F_1, F_2), \qquad i = 3, \cdots, k, \tag{15}$$

where $\sim$ denotes similarity (Section 2.3) and where, for $0 \leq j < d_2$, $S_j(F_1, F_2)$ is a polynomial of degree at most j, each of whose coefficients is a determinant of order $d_1 + d_2 - 2j$ with coefficients of F_1 and F_2 as its elements. In particular, $S_0(F_1, F_2)$ is the classical *resultant* [5, Ch. 12] of F_1 and F_2; in general, following Collins, we shall call $S_j(F_1, F_2)$ the jth *subresultant* of F_1 and F_2. The constants of similarity for (15) may be expressed as products of powers of $\alpha_3, \cdots, \alpha_i, \beta_3, \cdots, \beta_i, f_2, \cdots, f_i$, and (-1).

Brown and Traub [8] rederive these results in somewhat greater generality; their treatment is brief and simple, and shows clearly how the subresultants arise.

We shall now establish bounds on the coefficients of the subresultants

$$T_i = S_{d_{i-1}-1}(F_1, F_2), \qquad i = 3, \cdots, k. \tag{16}$$

Let

$$\begin{aligned} m_i &= \tfrac{1}{2}(d_1 + d_2 + 2) - d_{i-1} \\ &= [d_1 + d_2 - 2(d_{i-1} - 1)], \qquad i = 3, \cdots, k. \end{aligned} \tag{17}$$

This is an approximate measure of degree loss. As we proceed through the PRS, it increases monotonically from $m_3 \geq 1$ to $m_k \leq \frac{1}{2}(d_1 + d_2)$. In a normal PRS, m_i increases by 1 when i increases by 1; in general, the increment is δ_{i-1}. Since each coefficient of T_i is a determinant of order $2m_i$ with coefficients of F_1 and F_2 as its elements, our bounds depend simply on m_i.

If the coefficients of F_1 and F_2 are integers bounded in magnitude by c, then by Hadamard's theorem [1, p. 375] the coefficients of T_i are bounded in magnitude by

$$(2m_i c^2)^{m_i}. \tag{18}$$

Taking the logarithm (to the same base as the base of the number system), we see that the coefficients of T_i are bounded in length by

$$m_i[2l + \log(2m_i)], \tag{19}$$

where $l = \log c$ bounds the lengths of the coefficients of F_1 and F_2. Although the growth permitted by this bound is slightly faster than linear in m_i, the difference is rarely discernible since the second term is usually small compared to the first.

If the coefficients of F_1 and F_2 are polynomials in $y_1, \cdots, y_n$ over the integers, with degree at most e_j in y_j, then clearly the coefficients of T_i have degree at most

$$2m_i e_j \tag{20}$$

in y_j, for $j = 1, \cdots, n$. Thus, the growth in degree is strictly linear in m_i. If the polynomial coefficients of F_1 and F_2 have at most t terms each and have integer coefficients bounded in magnitude by c, then the integer coefficients of the polynomial coefficients of T_i are bounded in magnitude by

$$(2m_i c^2 t^2)^{m_i}. \tag{21}$$

This generalization of (18) follows easily from the results of [9]. Taking the logarithm, we see that these integer coefficients are bounded in length by

$$m_i[2l + \log (2m_i) + 2 \log t], \tag{22}$$

which is a generalization of (19).

In the case of a primitive PRS, (15) and (16) imply that $F_i \mid T_i$, so the bounds (18) and (20) on the coefficients of T_i also apply to the coefficients of F_i. When the coefficients of F_i and T_i are polynomials, it may happen in rare cases (see Section 5.2) that one or more integer coefficients of polynomial coefficients of F_i are larger than any integer coefficient of any polynomial coefficient of T_i. Nevertheless, we conjecture that no integer coefficient of any polynomial coefficient of F_i can ever exceed the bound (21).

To justify the reduced PRS algorithm (Section 3.4), it is shown in [7] and [8] that $T_i \mid F_i$ for $i = 3, \cdots, k$, and therefore all coefficients of F_i are in $\mathfrak{g}$. In the case of a normal reduced PRS, it is further shown that

$$F_i = \pm T_i, \qquad i = 3, \cdots, k, \tag{23}$$

and therefore the bounds (18)–(22) apply to the coefficients of F_i in this case as well.

3.6 THE SUBRESULTANT PRS ALGORITHM. If we could choose the β_i in such a way as to satisfy (23) even in abnormal cases, then no coefficient GCD's would be needed to compute the F_i, and yet the bounds (18)–(22) would apply to their coefficients. Fortunately this can be done. It is shown in [8] that $F_i = T_i$ for $i = 3, \cdots, k$ provided that we choose

$$\begin{aligned} \beta_3 &= (-1)^{\delta_1+1}, \\ \beta_i &= -f_{i-2}\psi_i^{\delta_{i-2}}, \qquad i = 4, \cdots, k, \end{aligned} \tag{24}$$

where

$$\begin{aligned} \psi_3 &= -1, \\ \psi_i &= (-f_{i-2})^{\delta_{i-3}}\psi_{i-1}^{1-\delta_{i-3}}, \qquad i = 4, \cdots, k. \end{aligned} \tag{25}$$

At the present time it is not known whether or not these equations imply $\psi_i, \beta_i \in \mathfrak{g}$. In any event, ψ_i and β_i belong to the quotient field $\mathfrak{F}$ of $\mathfrak{g}$, and they yield the PRS, $F_1, F_2, T_3, \cdots, T_k$ in $\mathfrak{g}[x]$.

It is possible to eliminate ψ_i from (24), and write β_i explicitly as a product of powers of $f_2, \cdots, f_{i-2}$, and (-1). The resulting formula is given in [7].

In the case of a normal PRS, note that the subresultant PRS algorithm (24) and the reduced PRS algorithm (13) agree up to signs as required by (23); in fact, the signs also agree if δ_1 is odd.

It is natural to wonder how close a subresultant PRS is likely to be to the corresponding primitive PRS. In the example (4), the subresultant PRS is

$$1, 0, 1, 0, -3, -3, 8, 2, -5$$

$$3, 0, 5, 0, -4, -9, 21$$

$$15, 0, -3, 0, 9$$

$$\begin{gathered}65,\ 125,\ -245\\ 9326,\ -12300\\ 260708\end{gathered} \tag{26}$$

which differs very little from the corresponding primitive PRS (12), except at the last step.

Although it is not known whether this behavior is typical, it is known that there is a broad spectrum of possibilities. On the one hand, there are subresultant PRS's in which all polynomials except the last are primitive. On the other hand [1, p. 377], let $F_1, F_2, \cdots, F_k$ be a normal subresultant PRS, and let G be a primitive polynomial with leading coefficient g. Then it is easy to show that the subresultant PRS for GF_1 and GF_2 is $GF_1, GF_2, g^{\delta_1+1}GF_3, g^{\delta_1+3}GF_4, \cdots, g^{\delta_1+2k-5}GF_k$, which diverges linearly from the primitive PRS $GF_1, GF_2, \cdots, GF_k$.

3.7 FURTHER IMPROVEMENTS. Having used the subresultant PRS algorithm to compute $F_i = T_i$ [see (16)] for $i = 3, \cdots, j$, it may happen that a divisor τ_j of $\operatorname{cont}(T_j)$ is available with little or no extra work. For example, we know that $\bar{g} = \gcd(f_1, f_2)$ divides T_j for $j = 3, \cdots, k$, and it occasionally happens that f_i divides T_j for some $i < j$.

When such a τ_j is available, we would like to incorporate it into β_j, so that $F_j = T_j/\tau_j$. However, if we do so, we cannot necessarily complete the PRS by direct application of (24). Fortunately, it is possible to modify (24) so as to complete the PRS and furthermore to guarantee that $F_i \mid T_i$ for $i = j + 1, \cdots, k$.

Let $F_1, F_2, T_3, \ldots, T_k$ be a subresultant PRS, and let $F_1, F_2, F_3, \ldots, F_k$ be an *improved PRS* with $F_i = T_i/\tau_i$ for $i = 3, \cdots, k$. Let $T_1 = F_1$, $T_2 = F_2$, and $\tau_1 = \tau_2 = 1$, and let t_i denote the leading coefficient of T_i. Then

$$\begin{aligned} T_i &= \tau_i F_i,\\ t_i &= \tau_i f_i,\end{aligned} \tag{27}$$

where $i = 1, \cdots, k$. Now from (9), (24), (25), and (27) it is easy to show that the improved PRS is defined by

$$\begin{aligned} \beta_3/\tau_3 &= (-1)^{\delta_1+1},\\ \beta_i/\tau_i &= -f_{i-2}\psi_i^{\delta_{i-2}}\tau_{i-1}^{-\delta_{i-2}-1}, \qquad i = 4, \cdots, k,\end{aligned} \tag{28}$$

where

$$\begin{aligned} \psi_3 &= -1,\\ \psi_i &= (-\tau_{i-2}f_{i-2})^{\delta_{i-3}}\psi_{i-1}^{1-\delta_{i-3}}, \qquad i = 4, \cdots, k.\end{aligned} \tag{29}$$

Given $F_1, \cdots, F_{i-1}$, we first compute $\operatorname{prem}(F_{i-2}, F_{i-1})$ and divide by (β_i/τ_i) from (28) to obtain the subresultant $T_i = \tau_i F_i$. We then choose τ_i, and divide by it to obtain F_i.

Clearly, there are many possible variations on this theme. To illustrate the theme, let us return to the example (4), for which the primitive PRS is (12) and the subresultant PRS is (26). The reader should imagine the problem to be enough harder that he could not easily discover the contents of the subresultants T_i. Choosing $\tau_i = f_{i-1}$, whenever $f_{i-1} \mid T_i$, and $\tau_i = 1$ otherwise, the improved PRS is

$$1, 0, 1, 0, -3, -3, 8, 2, -5$$
$$3, 0, 5, 0, -4, -9, 21$$
$$5, 0, -1, 0, 3$$
$$13, 25, -49$$
$$9326, -12300$$
$$260708. \tag{30}$$

By this simple strategy, we have succeeded in making F_3 and F_4 primitive, but we have failed to remove the factor of 2 from F_5.

3.8 COMPARISON. Let us now compare the foregoing algorithms. It is clear that the Euclidean PRS algorithm suffers so severely from coefficient growth that it does not merit further consideration. We also exclude the reduced PRS algorithm because it is never better than the subresultant PRS algorithm, and it can be extremely inefficient in certain abnormal cases.

In comparing the primitive PRS algorithm and the subresultant PRS algorithm, we must compare the cost of computing the primitive part at each step with the advantage of having possibly smaller coefficients. In most cases, the primitive-part calculations represent a substantial fraction of the total effort in the primitive PRS algorithm, and there is little if any compensating advantage. However, there may conceivably be some cases in which the subresultant PRS diverges so far from the primitive PRS that the primitive PRS algorithm is actually faster.

4. The Modular Algorithm

4.1 INTRODUCTION. Let F be a nonzero polynomial with integer coefficients. Let f denote the leading coefficient of F, and let p be a prime which does not divide f. Then if F is irreducible over the integers modulo p, it is also irreducible over the integers. This fact has long been exploited by those interested in polynomial factoring.

Similarly, let F_1 and F_2 be nonzero polynomials with leading coefficients f_1 and f_2, and let p be a prime which does not divide f_1 or f_2. Then if F_1 and F_2 are relatively prime over the integers modulo p, they are relatively prime over the integers. Even if F_1 and F_2 are not relatively prime, the computation of their GCD over the integers modulo p may provide useful information concerning their GCD over the integers.

We shall develop this idea into a general algorithm (Section 4.3) for computing the GCD of univariate or multivariate polynomials over the integers.

4.2 BASIC CONCEPTS. Let $\mathcal{I}$ be a unique factorization domain in which GCD's can somehow be computed, and let $\mathcal{I}[x_1, \cdots, x_v]$ denote the domain of polynomials in $x_1, \cdots, x_v$ with coefficients in $\mathcal{I}$. When $v > 1$, we shall *not* view the elements of this domain as polynomials in x_1 with coefficients in $\mathcal{I}[x_2, \cdots, x_v]$. Instead, we shall generalize the concepts of Section 2 to apply directly to multivariate polynomials.

Let the *exponent vector* of a term be the vector of its exponents, and define the *lexicographical ordering* of exponent vectors $\mathbf{d} = (d_1, \cdots, d_v)$ and $\mathbf{e} = (e_1, \cdots, e_v)$ as follows. If $d_i = e_i$ for $i = 1, \cdots, v$, then $\mathbf{d} = \mathbf{e}$. Otherwise, let j be the smallest integer such that $d_j \neq e_j$. If $d_j < e_j$, then $\mathbf{d} < \mathbf{e}$, while if $d_j > e_j$, then $\mathbf{d} > \mathbf{e}$. Assuming that the terms of a polynomial F are arranged in lexicographically decreasing order of their exponent vectors, the first term is called the *leading term*. The coefficient of the leading term is called the *leading coefficient*, and is denoted by $\mathrm{lc}(F)$. The exponent vector of the leading term is called the *degree vector*, and is denoted by $\partial(F)$. Note that $\partial(FG) = \partial(F) + \partial(G)$.

A polynomial is called *primitive* if its nonzero coefficients are relatively prime; in particular, all polynomials over a field are primitive. The *content* and *primitive part* are defined exactly as in Section 2.4, and it is again true that the GCD of two polynomials is the product of the GCD of their contents and the GCD of their primitive parts.

4.3 ALGORITHM M. Let Z denote the domain of integers, and let Z_p denote the field of integers modulo a prime p. Let F_1' and F_2' be given nonzero polynomials in $Z[x_1, \cdots, x_v]$. Algorithm M computes their GCD, G', and their *cofactors*, $H_1' = F_1'/G'$ and $H_2' = F_2'/G'$, making essential use of Algorithm P (Section 4.5), which computes the GCD and cofactors of given nonzero polynomials in $Z_p[x_1, \cdots, x_v]$. Each of these algorithms obtains the cofactors for the essential purpose of verifying the GCD; however, they are frequently a very welcome byproduct of the computation.

Let $c_1 = \mathrm{cont}(F_1')$, $c_2 = \mathrm{cont}(F_2')$, and $c = \gcd(c_1, c_2)$. Also, let $F_1 = F_1'/c_1$, $F_2 = F_2'/c_2$, $G = \gcd(F_1, F_2)$, $H_1 = F_1/G$, and $H_2 = F_2/G$. Then $G' = cG$, $H_1' = (c_1/c)H_1$, and $H_2' = (c_2/c)H_2$.

Let f_1, f_2, g, h_1, and h_2 be the leading coefficients of F_1, F_2, G, H_1, and H_2, respectively. As in Section 2.4, define $\bar{g} = \gcd(f_1, f_2)$ and $\bar{G} = (\bar{g}/g)G$. Also, define $\bar{F}_1 = \bar{g}F_1$, $\bar{F}_2 = \bar{g}F_2$, $\bar{H}_1 = gH_1$, and $\bar{H}_2 = gH_2$, so that $\bar{F}_1 = \bar{G}\bar{H}_1$ and $\bar{F}_2 = \bar{G}\bar{H}_2$. Note that $\mathrm{pp}(\bar{G}) = G$, $\mathrm{lc}(\bar{G}) = \bar{g}$, $\mathrm{lc}(\bar{H}_1) = gh_1 = f_1$, and $\mathrm{lc}(\bar{H}_2) = gh_2 = f_2$. Observe that f_1, f_2, and $\bar{g}$ can easily be obtained at the beginning of the computation, while g cannot be known until the end.

Let $\mathbf{d} = \partial(G)$. Although $\mathbf{d}$ cannot be determined until the end of the computation, it is evident that $\mathbf{d} \leq \min(\partial(F_1), \partial(F_2))$.

At any given time in the execution of Algorithm M, there is a set of odd primes $p_1, \cdots, p_n$ that have been used and not discarded. Furthermore, for $i = 1, \cdots, n$, the algorithm has computed

$$\begin{aligned} \tilde{F}_1^{(i)} &= \bar{F}_1 \bmod p_i, \\ \tilde{F}_2^{(i)} &= \bar{F}_2 \bmod p_i, \end{aligned} \tag{31}$$

and three polynomials $\tilde{G}^{(i)}$, $\tilde{H}_1^{(i)}$, and $\tilde{H}_2^{(i)}$ satisfying

$$\tilde{G}^{(i)} = \tilde{g}^{(i)} \cdot \gcd(\tilde{F}_1^{(i)}, \tilde{F}_2^{(i)}) \tag{32}$$

and

$$\begin{aligned} \tilde{F}_1^{(i)} &= \tilde{G}^{(i)}\tilde{H}_1^{(i)}, \\ \tilde{F}_2^{(i)} &= \tilde{G}^{(i)}\tilde{H}_2^{(i)} \end{aligned} \tag{33}$$

in $Z_{p_i}[x_1, \cdots, x_v]$, where

$$\tilde{g}^{(i)} = \bar{g} \bmod p_i. \tag{34}$$

Since the GCD in (32) is unit normal (in this case, monic) by definition, it follows that $\mathrm{lc}(\tilde{G}^{(i)}) = \tilde{g}^{(i)}$.

Since $G \mid F_1$ and $G \mid F_2$ in $Z[x_1, \cdots, x_v]$, we have $G_{p_i} \mid \tilde{F}_1^{(i)}$, $G_{p_i} \mid \tilde{F}_2^{(i)}$, and therefore $G_{p_i} \mid \tilde{G}^{(i)}$ in $Z_{p_i}[x_1, \cdots, x_v]$, where $G_{p_i} = G \bmod p_i$. It follows immediately that

$\partial(\tilde{G}^{(i)}) \geq \partial(G_{p_i}) = \partial(G) = \mathbf{d}$. The algorithm keeps only those p_i for which $\partial(\tilde{G}^{(i)})$ is minimal to date; hence, we always have $\partial(\tilde{G}^{(i)}) = \mathbf{e} \geq \mathbf{d}$. If $\mathbf{e} > \mathbf{d}$, the algorithm will discover the fact (for proof, see Section 4.4) and start over. Eventually, $\mathbf{e} = \mathbf{d}$, and

$$\begin{aligned} \tilde{G}^{(i)} &\equiv \tilde{G} \bmod p_i, \\ \tilde{H}_1^{(i)} &\equiv \tilde{H}_1 \bmod p_i, \\ \tilde{H}_2^{(i)} &\equiv \tilde{H}_2 \bmod p_i, \end{aligned} \tag{35}$$

for $i = 1, \cdots, n$.

Instead of preserving all of the quadruples $(p_i, \tilde{G}^{(i)}, \tilde{H}_1^{(i)}, \tilde{H}_2^{(i)})$, we maintain only the integer

$$q = \prod_{i=1}^{n} p_i, \tag{36}$$

and the unique polynomials G^*, H_1^*, and H_2^* with integer coefficients of magnitude less than $q/2$, such that

$$\begin{aligned} G^* &\equiv \tilde{G}^{(i)} \bmod p_i, \\ H_1^* &\equiv \tilde{H}_1^{(i)} \bmod p_i, \\ H_2^* &\equiv \tilde{H}_2^{(i)} \bmod p_i, \end{aligned} \tag{37}$$

for $i = 1, \cdots, n$. Now as soon as $\mathbf{e} = \mathbf{d}$, we see from (35) and (37) that

$$\begin{aligned} G^* &\equiv \tilde{G} \bmod q, \\ H_1^* &\equiv \tilde{H}_1 \bmod q, \\ H_2^* &\equiv \tilde{H}_2 \bmod q. \end{aligned} \tag{38}$$

When we also achieve

$$q > \mu = 2 \max |\psi|, \tag{39}$$

where ψ ranges over the coefficients of $\tilde{G}$, $\tilde{H}_1$, and $\tilde{H}_2$, it follows that

$$\begin{aligned} G^* &= \tilde{G}, \\ H_1^* &= \tilde{H}_1, \\ H_2^* &= \tilde{H}_2. \end{aligned} \tag{40}$$

To obtain the final results, we then use the relations

$$\begin{aligned} G &= \mathrm{pp}(\tilde{G}), \\ H_1 &= \tilde{H}_1/g, \\ H_2 &= \tilde{H}_2/g, \end{aligned} \tag{41}$$

and

$$\begin{aligned} G' &= cG, \\ H_1' &= (c_1/c)H_1, \\ H_2' &= (c_2/c)H_2, \end{aligned} \tag{42}$$

all of which were derived earlier.

In order to guarantee that the coefficients of G^*, H_1^*, and H_2^* converge to the correct signed integers, we shall assume throughout that the integers modulo r (where r is either q or one of the p_i) are represented as integers of magnitude less than $r/2$. The algorithm follows:

(1) Set $c_1 = \text{cont}(F_1')$, $c_2 = \text{cont}(F_2')$, $c = \gcd(c_1, c_2)$.
(2) Set $F_1 = F_1'/c_1$, $F_2 = F_2'/c_2$.
(3) Set $f_1 = \text{lc}(F_1)$, $f_2 = \text{lc}(F_2)$, $\bar{g} = \gcd(f_1, f_2)$.
(4) Set $n = 0$, $\mathbf{e} = \min(\partial(F_1), \partial(F_2))$.
(5) Set $\bar{\mu} = 2\bar{g} \max |\varphi|$, where φ ranges over the coefficients of F_1 and F_2.

Usually, it will be true that $\bar{\mu} > \mu$, but exceptions are possible as discussed in Section 5.2.

(6) Let p be a new odd prime not dividing f_1 or f_2.

(7) Set $\tilde{g} = \bar{g} \bmod p$, $\tilde{F}_1 = \tilde{g}F_1 \bmod p$, $\tilde{F}_2 = \tilde{g}F_2 \bmod p$.

(8) Invoke Algorithm P (Section 4.5) to compute $\tilde{G} = \tilde{g}\cdot\gcd(\tilde{F}_1, \tilde{F}_2)$, $\tilde{H}_1 = \tilde{F}_1/\tilde{G}$, and $\tilde{H}_2 = \tilde{F}_2/\tilde{G}$, all in $Z_p[x_1, \cdots, x_v]$. These relations imply that $\text{lc}(\tilde{G}) = \tilde{g}$, and $\partial(\tilde{G}) \geq \mathbf{d}$.

(9) If $\partial(\tilde{G}) = 0$, set $G = 1$, $H_1 = F_1$, $H_2 = F_2$, and skip to Step (15). If $\partial(\tilde{G}) > \mathbf{e}$, go back to step 6. If $\partial(\tilde{G}) < \mathbf{e}$, set $n = 0$, $\mathbf{e} = \partial(\tilde{G})$.

(10) Set $n = n + 1$.

(11) If $n = 1$, set $q = p$, $G^* = \tilde{G}$, $H_1^* = \tilde{H}_1$, $H_2^* = \tilde{H}_2$. Otherwise, update the quadruple (q, G^*, H_1^*, H_2^*) to include $(p, \tilde{G}, \tilde{H}_1, \tilde{H}_2)$ by using the Chinese remainder algorithm (Section 4.8) with moduli $m_1 = q$ and $m_2 = p$ to extend (37) (coefficient by coefficient), and then replacing q by pq to extend (36).

(12) If $q < \bar{\mu}$, go back to Step (6). Otherwise, we now know that (40) holds unless $\mathbf{e} > \mathbf{d}$ or $q < \mu$. To exclude these unlikely possibilities, it suffices to prove the relations $G^*H_1^* = \bar{F}_1$ and $G^*H_2^* = \bar{F}_2$, which hold modulo q by (31), (33), (36), and (37).

(13) Choose μ^* such that $\mu^*/2$ is an integer bound on the magnitudes of the coefficients of $G^*H_1^*$ and $G^*H_2^*$. If $q < \mu^*$, go back to Step (6). Otherwise, we have $q \mid (G^*H_1^* - \bar{F}_1)$ and $q > \max(\mu^*, \bar{\mu}) \geq \max |\varphi|$, where φ ranges over the coefficients of $(G^*H_1^* - \bar{F}_1)$, and therefore $G^*H_1^* = \bar{F}_1$. Similarly, $G^*H_2^* = \bar{F}_2$, and therefore (40) is established.

(14) Set $G = \text{pp}(G^*)$, $g = \text{lc}(G)$, $H_1 = H_1^*/g$, $H_2 = H_2^*/g$.

(15) Set $G' = cG$, $H_1' = (c_1/c)H_1$, $H_2' = (c_2/c)H_2$, and return.

4.4 UNLUCKY PRIMES. We shall call the prime p chosen in Step (6) of Algorithm M *lucky* if $\mathbf{e} = \mathbf{d}$, and *unlucky* otherwise.

In executing Algorithm M, the first lucky prime causes us to discard the information from any previous unlucky primes [by setting $n = 0$ in the third part of Step (9)], and to set $\mathbf{e} = \mathbf{d}$. Any subsequent unlucky primes are rejected in the second part of Step (9). In Step (13), G^*, H_1^*, and H_2^* are rejected if no lucky primes have yet been encountered, or if q is still less than μ.

In Theorem 1, we shall prove that all of the unlucky primes are divisors of an integer σ, which depends only on F_1 and F_2. Using this result, Theorem 2 bounds the number of unlucky primes which might occur, thereby establishing the fact that the algorithm terminates. Finally, Theorem 3 shows that the probability of p being unlucky is at most v/p.

In practice, this probability is always exceedingly small, since p is chosen in the interval

$$\alpha = \tfrac{1}{2}(\beta + 1) < p \leq \beta, \tag{43}$$

where β is the largest integer that fits in a single machine word. Thus, if F_1 and F_2 are relatively prime, we can expect to prove it with only a single prime. Otherwise, the expected number of primes to determine the GCD and the cofactors is $\bar{n} = \log_\alpha \mu^{**}$, where μ^{**} is the final value of μ^* in Step (13), and it can be shown [see (94)] that $\bar{n}$ rarely exceeds $4l + 2$ where l is the length (to the base α) of the longest coefficient in F_1 or F_2.

THEOREM 1. *Let F_1 and F_2 be given nonzero primitive polynomials in $Z[x_1, \cdots, x_v]$, let $G = \gcd(F_1, F_2)$, and let $d_i = \partial_i(G)$, where ∂_i denotes the degree in X_i. Also, let $S_j^{(i)}(F_1, F_2)$ denote the jth subresultant of F_1 and F_2 viewed as univariate polynomials in x_i with polynomial coefficients, and let σ_i be the (integer) content of $S_{d_i}^{(i)}(F_1, F_2)$ viewed as a multivariate polynomial with integer coefficients. (Here d_i is the degree of G in X_i, and not, as in Section 3.5, the degree of the ith polynomial in a PRS.) Finally, let*

$$\sigma = \prod_{i=1}^{v} \sigma_i. \tag{44}$$

Then every unlucky prime divides σ.

PROOF. For fixed p from Step (6) of Algorithm M, let $\tilde{F}_1 = F_1 \bmod p$ and $\tilde{F}_2 = F_2 \bmod p$. Also, let $\tilde{G} = \gcd(\tilde{F}_1, \tilde{F}_2)$ in $Z_p[x_1, \cdots, x_v]$, and let $e_i = d_i(\tilde{G})$.

If p is unlucky, then there is some i with $d_i < e_i$, and therefore $S_{d_i}^{(i)}(\tilde{F}_1, \tilde{F}_2) \equiv 0 \bmod p$. Now let c_1 and c_2 denote the leading (polynomial) coefficients of F_1 and F_2, respectively, relative to x_i. If $p \nmid c_1c_2$, then $S_{d_i}^{(i)}(F_1, F_2) \equiv S_{d_i}^{(i)}(\tilde{F}_1, \tilde{F}_2) \bmod p$. Otherwise, suppose p divides the first k leading coefficients of F_1 for some $k > 0$. Then it can be shown that $S_{d_i}^{(i)}(F_1, F_2) \equiv c_2^k S_{d_i}^{(i)}(\tilde{F}_1, \tilde{F}_2) \bmod p$. In either case $S_{d_i}^{(i)}(F_1, F_2) \equiv 0 \bmod p$, $p \mid S_{d_i}^{(i)}(F_1, F_2)$, $p \mid \sigma_i$, and finally $p \mid \sigma$, as was to be shown.

THEOREM 2. *Let u be the number of unlucky primes $p > \alpha$, where $\alpha \geq 2$ is a given integer. Let*

$$c = \max |\varphi|, \tag{45}$$

where φ ranges over the coefficients of F_1 and F_2. Let

$$m = \tfrac{1}{2} \max_{i=1}^{v} (\partial_i(F_1) + \partial_i(F_2)). \tag{46}$$

Let

$$t = \max_{i=1}^{v} t_i, \tag{47}$$

where t_i is the maximum number of terms in any polynomial coefficient of F_1 or F_2 viewed as univariate polynomials in x_i. Finally, let

$$\bar{u} = mv \log_\alpha (2mc^2t^2). \tag{48}$$

Then $u < \bar{u}$.

PROOF. Let P be the product of the unlucky primes $p > \alpha$. Since $P \mid \sigma$ by Theorem 1, we have $\alpha^u < P \leq \sigma$. Now by (21), $\sigma_i \leq (2mc^2t^2)^m$, so $\sigma \leq (2mc^2t^2)^{mv}$. It follows that $u < \log_\alpha \sigma \leq \bar{u}$, as was to be shown.

THEOREM 3. *The probability that p is unlucky is at most v/p.*

PROOF. For all sufficiently small problems, we have $\sigma_i < p$ for $i = 1, \cdots, v$, and therefore p cannot be unlucky. However, in the general case, we may assume that the quantities $\sigma_i \bmod p$ are independent random variables in Z_p. Hence the probability that $p \mid \sigma_i$ is p^{-1}, and the probability that $p \mid \sigma$ is $1 - (1 - p^{-1})^v \leq vp^{-1}$, where this last inequality follows by induction on v. This completes the proof.

4.5 ALGORITHM P. Let F_1' and F_2' be given nonzero polynomials in $Z_p[x_1, \cdots, x_v]$, where p is a fixed, sufficiently large prime. [If p is too small, the elements of Z_p may be exhausted by Step (6) of the algorithm.] Algorithm P computes $G' = \gcd(F_1', F_2')$, $H_1' = F_1'/G'$, and $H_2' = F_2'/G'$. Since Z_p is a field, F_1' and F_2' are automatically primitive, and G' is monic.

In the univariate case, Algorithm P simply invokes Algorithm U, which is presented in Section 4.7.

In the multivariate case, we could single out a main variable, and then apply Algorithm C. If so, we would still be faced with the problem of coefficient growth, since the polynomial coefficients would grow in degree even though their integer coefficients (in Z_p) could not grow in size.

To avoid coefficient growth altogether, we again use the idea of Algorithm M. Instead of solving the given problem directly, we solve one or more related problems in $Z_p[x_1, \cdots, x_{v-1}]$, and then reconstruct the desired result.

Let F_1' and F_2' be viewed as polynomials in $\mathfrak{s}[x_1, \cdots, x_{v-1}]$, where $\mathfrak{s}$ denotes the polynomial domain $Z_p[x_v]$. Although F_1' and F_2' are primitive as elements of $Z_p[x_1, \cdots, x_v]$, they need not be primitive as elements of $\mathfrak{s}[x_1, \cdots, x_{v-1}]$. Let c_1, c_2, c, $F_1, F_2, G, H_1, H_2, f_1, f_2, g, h_1, h_2, \bar{g}, \bar{G}, \bar{F}_1, \bar{F}_2, \bar{H}_1$, and $\bar{H}_2$ be defined as in Algorithm M. Now, however, all of the lower case symbols denote polynomials in $\mathfrak{s} = Z_p[x_v]$, while the upper case symbols denote polynomials in $\mathfrak{s}[x_1, \cdots, x_{v-1}]$.

For any fixed $b \in Z_p$, let $\mathfrak{s}_b$ denote the field of polynomials in $\mathfrak{s}$ modulo the irreducible polynomial $(x_v - b)$. Since for polynomials $f \in \mathfrak{s}$, the quantity $f(x_v) \bmod (x_v - b)$ is equal to $f(b) \in Z_p$, we see that $\mathfrak{s}_b$ is precisely Z_p.

Algorithm P is essentially identical to Algorithm M, except that v is replaced by $v - 1$, Z is replaced by $\mathfrak{s}$, the prime $p \in Z$ is replaced by the irreducible polynomial $(x_v - b) \in \mathfrak{s}$, and Z_p is replaced by $\mathfrak{s}_b = Z_p$.

In this situation, (31) is replaced by

$$\begin{aligned} \tilde{F}_1^{(i)} &= \bar{F}_1 \bmod (x_v - b_i), \\ \tilde{F}_2^{(i)} &= \bar{F}_2 \bmod (x_v - b_i), \end{aligned} \tag{49}$$

while the polynomials $\tilde{G}^{(i)}$, $\tilde{H}_1^{(i)}$, and $\tilde{H}_2^{(i)}$ satisfy (32) and (33) in $\mathfrak{s}_{b_i}[x_1, \cdots, x_{v-1}] = Z_p[x_1, \cdots, x_{v-1}]$. Furthermore, (34) becomes

$$\tilde{g}^{(i)} \equiv \bar{g} \bmod (x_v - b_i), \tag{50}$$

and when $\mathbf{e} = \mathbf{d}$, we have

$$\begin{aligned} \tilde{G}^{(i)} &\equiv \bar{G} \bmod (x_v - b_i), \\ \tilde{H}_1^{(i)} &\equiv \bar{H}_1 \bmod (x_v - b_i), \\ \tilde{H}_2^{(i)} &\equiv \bar{H}_2 \bmod (x_v - b_i), \end{aligned} \tag{51}$$

for $i = 1, \cdots, n$, in place of (35). Also, (36) becomes

$$q = \prod_{i=1}^{n} (x_v - b_i), \tag{52}$$

while G^*, H_1^*, and H_2^* [see (37)] are the unique polynomials (in $\mathcal{I}[x_1, \cdots, x_{v-1}]$) with coefficients (in $\mathcal{I}$) of degree (in x_v) less than n, such that

$$\begin{aligned} G^* &\equiv G^{(i)} \bmod (x_v - b_i), \\ H_1^* &\equiv H_1^{(i)} \bmod (x_v - b_i), \\ H_2^* &\equiv H_2^{(i)} \bmod (x_v - b_i), \end{aligned} \tag{53}$$

for $i = 1, \cdots, n$. Now as soon as $\mathbf{e} = \mathbf{d}$, we see from (51) and (53) that (38) holds. When we also achieve

$$n > \nu = \max(\partial_v(\bar{G}), \partial_v(\bar{H}_1), \partial_v(\bar{H}_2)), \tag{54}$$

where ∂_v denotes the degree in x_v, it follows that (40) holds. To obtain the final results, we then use (41) and (42) as in Algorithm M.

Although the preceding discussion is sufficient in principle to define Algorithm P, the interested reader may find it instructive to compare the following detailed description with the earlier presentation (Section 4.3) of Algorithm M.

(1) If $v = 1$, then F_1' and F_2' are elements of $\mathcal{I}$; invoke Algorithm U to comput $G' = \gcd(F_1', F_2')$, and return. Otherwise use Algorithm U to compute $c_1 = \text{cont}(F_1')$ $c_2 = \text{cont}(F_2')$, $c = \gcd(c_1, c_2)$.

(2) Set $F_1 = F_1'/c_1$, $F_2 = F_2'/c_2$.

(3) Set $f_1 = \text{lc}(F_1)$, $f_2 = \text{lc}(F_2)$, $\bar{g} = \gcd(f_1, f_2)$.

(4) Set $n = 0$, $\mathbf{e} = \min(\boldsymbol{\partial}(F_1), \boldsymbol{\partial}(F_2))$.

(5) Set $\bar{\nu}_1 = \partial_v(\bar{g}) + \partial_v(F_1)$, $\bar{\nu}_2 = \partial_v(\bar{g}) + \partial_v(F_2)$, $\bar{\nu} = \max(\bar{\nu}_1, \bar{\nu}_2)$. It follows that $\bar{\nu}_1 = \partial_v(\bar{F}_1) = \partial_v(\bar{G}) + \partial_v(\bar{H}_1)$, $\bar{\nu}_2 = \partial_v(\bar{F}_2) = \partial_v(\bar{G}) + \partial_v(\bar{H}_2)$, and $\bar{\nu} \geq \nu$.

(6) Let b be a new element of Z_p such that $(x_v - b) \nmid f_1 f_2$. If Z_p is exhausted, then p is too small and the algorithm fails.

(7) Set $\tilde{g} = \bar{g} \bmod (x_v - b)$, $\tilde{F}_1 = \bar{g}F_1 \bmod (x_v - b)$, $\tilde{F}_2 = \bar{g}F_2 \bmod (x_v - b)$.

(8) Invoke Algorithm P recursively to compute $\tilde{G} = \tilde{g} \cdot \gcd(\tilde{F}_1, \tilde{F}_2)$, $\tilde{H}_1 = \tilde{F}_1/\tilde{G}$, and $\tilde{H}_2 = \tilde{F}_2/\tilde{G}$, all in $\mathcal{I}_b[x_1, \cdots, x_{v-1}] = Z_p[x_1, \cdots, x_{v-1}]$. These relations imply that $\text{lc}(\tilde{G}) = \tilde{g}$, and $\boldsymbol{\partial}(\tilde{G}) \geq \mathbf{d}$.

(9) If $\boldsymbol{\partial}(\tilde{G}) = 0$, set $G = 1$, $H_1 = F_1$, $H_2 = F_2$, and skip to Step (15). If $\boldsymbol{\partial}(\tilde{G}) > \mathbf{e}$, go back to Step (6). If $\boldsymbol{\partial}(\tilde{G}) < \mathbf{e}$, set $n = 0$, $\mathbf{e} = \boldsymbol{\partial}(\tilde{G})$.

(10) Set $n = n + 1$.

(11) If $n = 1$, set $q = p$, $G^* = \tilde{G}$, $H_1^* = \tilde{H}_1$, $H_2^* = \tilde{H}_2$. Otherwise, update the quadruple (q, G^*, H_1^*, H_2^*) to include $(p, \tilde{G}, \tilde{H}_1, \tilde{H}_2)$ by using the Chinese remainder algorithm (Section 4.8) (which in this case is a form of interpolation [1, p. 430]) with moduli $m_1 = q$ and $m_2 = x_v - b$ to extend (53) (coefficient by coefficient), and then replacing q by $q(x_v - b)$ to extend (52).

(12) If $n \leq \bar{\nu}$, go back to Step (6). Otherwise, we now know that $n > \bar{\nu} \geq \nu$, so (40) holds unless $\mathbf{e} > \mathbf{d}$. To exclude this unlikely possibility, it suffices to prove the relations $G^*H_1^* = \bar{F}_1$ and $G^*H_2^* = \bar{F}_2$, which hold modulo q by (33), (49), (52), and (53).

(13) If $\partial_v(G^*) + \partial_v(H_1^*) \neq \bar{\nu}_1$ or $\partial_v(G^*) + \partial_v(H_2^*) \neq \bar{\nu}_2$, set $n = 0$ and go back to Step (6). Otherwise we have $q \mid (G^*H_1^* - \bar{F}_1)$ and $\partial_v(q) = n > \bar{\nu} \geq \bar{\nu}_1 \geq \partial_v(G^*H_1^* - \bar{F}_1)$, and therefore $G^*H_1^* = \bar{F}_1$. Similarly, $G^*H_2^* = \bar{F}_2$, and therefore (40) is established.

(14) Set $G = pp(G^*)$, $g = \text{lc}(G)$, $H_1 = H_1^*/g$, $H_2 = H_2^*/g$.

(15) Set $G' = cG$, $H_1' = (c_1/c)H_1$, $H_2' = (c_2/c)H_2$, and return.

4.6 UNLUCKY b-VALUES. We shall call the integer $b \in Z_p$ chosen in Step (6) of Algorithm P *lucky* if $\mathbf{e} = \mathbf{d}$, and *unlucky* otherwise.

In Theorem 4 we shall prove that all of the unlucky b-values are roots (in Z_p) of a polynomial σ (in $\mathfrak{s} = Z_p[x_v]$), which depends only on F_1 and F_2. Using this result, Theorem 5 bounds the total number u of unlucky b-values, thereby establishing the fact that the algorithm terminates.

If b is chosen at random from the elements of Z_p, the probability of its being unlucky is u/p. If p is large (as suggested in Section 4.4), this probability is exceedingly small. Thus if F_1 and F_2 are relatively prime, we can expect to prove it with only a single b-value. Otherwise the expected number of b-values to determine the GCD and the cofactors is $\bar{n} = \bar{\nu} + 1 = \max(\partial_v(\bar{F}_1), \partial_v(\bar{F}_2)) + 1$.

THEOREM 4. *Let F_1 and F_2 be given nonzero polynomials in $\mathfrak{s}[x_1, \cdots, x_{v-1}]$, where $\mathfrak{s} = Z_p[x_v]$. Let $G = \gcd(F_1, F_2)$, and let $d_i = \partial_i(G)$. Also, let $S_j^{(i)}(F_1, F_2)$ denote the jth subresultant of F_1 and F_2 viewed as univariate polynomials in x_i, with coefficients in $\mathfrak{s}[x_1, \cdots, x_{i-1}, x_{i+1}, \cdots, x_{v-1}]$, and let σ_i be the content (in $\mathfrak{s}$) of $S_{d_i}^{(i)}(F_1, F_2)$ viewed as a polynomial in $\mathfrak{s}[x_1, \cdots, x_{v-1}]$. (Here d_i is the degree of G in X_i, and not, as in Section 3.5, the degree of the ith polynomial in a PRS.) Finally, let*

$$\sigma = \prod_{i=1}^{v-1} \sigma_i. \tag{55}$$

Then every unlucky b-value is a root (in Z_p) of σ.

PROOF. The proof is analogous to the proof of Theorem 1.

THEOREM 5. *Let u be the total number of unlucky b-values, let*

$$m = \tfrac{1}{2} \max_{i=1}^{v-1} (\partial_i(F_1) + \partial_i(F_2)), \tag{56}$$

and let

$$e = \max(\partial_v(F_1), \partial_v(F_2)). \tag{57}$$

Finally, let

$$\bar{u} = 2me(v - 1). \tag{58}$$

Then $u \leq \bar{u}$.

PROOF. Let $P = \prod (x_v - b)$, where the product is taken over all unlucky b-values. Since $P \mid \sigma$ by Theorem 4, we have $u = \partial_v(P) \leq \partial_v(\sigma) = \sum \partial_v(\sigma_i)$. But by (20), $\partial_v(\sigma_i) \leq 2me$, so $u \leq 2me(v - 1)$, as was to be shown.

4.7 ALGORITHM U. Let F_1 and F_2 be given nonzero polynomials in $Z_p[x]$, where p is a fixed prime. Algorithm U computes $G = \gcd(F_1, F_2)$. Since Z_p is a field, G must be monic in order to satisfy the requirement of unit normality. Also since Z_p is a field, we may use the rational algorithm of Section 2.2.

In the example (4) with $p = 13$, the monic PRS which mirrors (6) is

$$\begin{array}{l} 1, 0, 1, 0, -3, -3, -5, 2, -5 \\ 1, 0, 6, 0, 3, -3, -6 \\ 1, 0, 5, 0, -2 \\ 1, -3 \\ 1. \end{array} \tag{59}$$

This proves that F_1 and F_2 are relatively prime in $Z_{13}[x]$, and hence also in $Z[x]$. Note that the quadratic and linear polynomials in (6) have coalesced in (59), because 13 divides the leading coefficient, 65, of the subresultant $S_2(F_1, F_2)$, which we computed in (26).

4.8 THE CHINESE REMAINDER ALGORITHM. We shall now present the Chinese remainder algorithm [1, pp. 253–254], which is used in Algorithm M to construct integers from their images modulo $p_1, \cdots, p_n$, and in Algorithm P to construct polynomials in $Z_p[x]$ from their images modulo $x - b_1, \cdots, x - b_n$. Although the algorithm may be used in any Euclidean domain, we shall simply state it for the domain of integers, and indicate parenthetically how it must be modified for the polynomial domain $\mathfrak{F}[x]$ where $\mathfrak{F}$ is any field.

Let m_1 and m_2 be relatively prime positive integers (monic polynomials). We shall call these the moduli; for efficiency they should be ordered so that $m_1 > m_2$ $[\partial(m_1) \geq \partial(m_2)]$. If u_1 and u_2 are given integers (polynomials), then there is a unique integer (polynomial) u such that $u \equiv u_1 \bmod m_1$, $u \equiv u_2 \bmod m_2$, and $0 \leq u < m_1m_2$ $[\partial(u) < \partial(m_1, m_2)]$.

To prove uniqueness, suppose u and u' both satisfy the conditions. Then $u \equiv u' \bmod m_1$ and $u \equiv u' \bmod m_2$. Since m_1 and m_2 are relatively prime, it follows that $m_1m_2 \mid (u - u')$, and therefore $u = u'$.

In the following algorithm for finding u, it is understood that $r = a \bmod b$ satisfies $0 \leq r < b$ $[\partial(r) < \partial(b)]$, and we assume without loss of generality that $0 \leq u_1 < m_1$ $[\partial(u_1) < \partial(m_1)]$ and $0 \leq u_2 < m_2$ $[\partial(u_2) < \partial(m_2)]$.

(1) Use the extended Euclid's algorithm (Section 1.4) to obtain an integer (polynomial) c such that $cm_1 \equiv 1 \bmod m_2$.

(2) Set $v = c(u_2 - u_1) \bmod m_2$. Note that $m_1 v \equiv u_2 - u_1 \bmod m_2$.

(3) Set $u = m_1v + u_1$. Clearly this satisfies all the requirements.

5. *Computing Time*

5.1 INTRODUCTION. In this section we shall study the computing times for Algorithm C (Section 2.4) augmented by the subresultant PRS algorithm (Section 3.6), for Algorithm M (Section 4.3), and for Algorithm P (Section 4.5) which supports Algorithm M. In particular, we shall develop asymptotic bounds on the maximum computing times C, M, and P, respectively, of these algorithms.

In deriving these bounds we shall assume that all arithmetic operations on integers and polynomials are performed by the classical algorithms [1]. Our purpose is to emphasize the superiority of the modular techniques to the classical ones. However, this superiority is established only when the given polynomials are sufficiently large and sufficiently dense. In the sparse case (many missing terms), Algorithm C will almost certainly benefit more than Algorithms M and P, but the gain is difficult to analyze.

Strictly speaking, our bounds do not apply to the worst cases, since they depend on the following assumptions:

(A1) In Algorithm C, it is assumed that abnormal PRS's do not occur at any level of recursion. A larger bound not depending on this assumption, can easily be derived by the same methods.

(A2) In Algorithms M and P, it is assumed that unlucky primes and unlucky b-values, respectively, do not occur.

(A3) In Algorithms C and M, it is assumed that the integer-length (Section 5.2) of an exact quotient of polynomials does not exceed the integer-length of the dividend. In Algorithm C, it is further assumed that the integer-length of a sum or difference of polynomials is the maximum of the integer-lengths of the summands, and that the integer-length of a product does not exceed the sum of the integer-lengths of the factors.

(A4) In Algorithm M, it is assumed that the integer-lengths of the given polynomials are small compared to the largest single-word integer β so that the supply of single-word primes will not be exhausted. Similarly it is assumed that the number of terms in the GCD is small compared to β, so that the required number of primes can be bounded in a simple and realistic way.

After introducing some basic concepts in Section 5.2, we shall discuss integer operations in Section 5.3, polynomial operations in Section 5.4, Algorithm C in Section 5.5, Algorithm P in Section 5.6, and Algorithm M in Section 5.7. Finally we compare Algorithms C and M in Section 5.8.

5.2 BASIC CONCEPTS. Let f and g be real functions defined on a set S. If there is a positive real number c such that $|f(x)| \leq c|g(x)|$ for all $x \in S$, we write $f \precsim g$, and say that f is *dominated* by g. If $f \precsim g$ and $g \precsim f$, we write $f \sim g$, and say that f and g are *codominant*. Finally, if $f \precsim g$ but $f \nsim g$, we write $f \prec g$, and say that f is *strictly dominated* by g. Clearly codominance is an equivalence relation among the real functions on S, while strict dominance defines a partial ordering among the resulting codominance classes. In the author's opinion, this notation and terminology (from [10]) are significantly superior to the traditional "little-oh" and "big-oh" notation [11, p. 5].

Since the purpose of the analysis is to provide insight, not detail, we shall consider only a minimum number of independent parameters. Every polynomial F which is considered in the analysis will be characterized either by its dimension vector (l, d), defined below, or by its degree in the main variable together with a single dimension vector for all its coefficients.

First we define the *length* of a nonzero integer to be the logarithm (to some fixed base such as 2 or 10) of its magnitude. Next we define the *integer-length* of a nonzero polynomial $F \in Z[x_1, \cdots, x_v]$ to be the maximum of the lengths of its nonzero coefficients; this will be denoted by $\mathrm{il}(F)$. Finally, for nonzero $F \in Z[x_1, \cdots, x_v]$ we define the *dimension vector* (l, d) by the relations $l = \mathrm{il}(F)$ and $d = \max\ (\partial_i(F))$.

For integers it is clear that the length of a product is the sum of the lengths of the factors. For polynomials, the integer-length of a product may be smaller or larger than the sum of the integer-lengths of the factors, because of the additive combination of terms in polynomial multiplication. For example, if $A(x) = x - 1$, then $A(x)^2 = x^2 - 2x + 1$, and $\mathrm{il}(A^2) = \log 2 > 2\,\mathrm{il}(A) = 0$. On the other hand, letting $B(x) = x^8 + 2x^7 + 3x^6 + 4x^5 + 5x^4 + 4x^3 + 3x^2 + 2x + 1$, we have $A(x)B(x) = x^9 + x^8 + x^7 + x^6 + x^5 - x^4 - x^3 - x^2 - x - 1$, and $\mathrm{il}(AB) = 0 < \mathrm{il}(A) + \mathrm{il}(B) = \log 5$. In this latter example, A acts as a first difference operator. The effect may be exhibited more dramatically by taking $A(x) = (x - 1)^n$ for some $n > 1$, and choosing B so that the nth differences of its coefficients are all equal to ± 1.

In spite of these exceptions, the author believes that assumption (A3) will lead to simpler and more realistic bounds than would otherwise be attainable.

5.3 INTEGER OPERATIONS. Let $T_I(\mathrm{op})$ denote the maximum computing time for the integer operation op, and let x_i denote an integer of length $l_i > 0$.

For addition and subtraction it is easy to show that

$$T_I(x_3 \leftarrow x_1 \pm x_2) \sim l_1 + l_2, \tag{60}$$

while for classical multiplication

$$T_I(x_3 \leftarrow x_1x_2) \sim l_1l_2. \tag{61}$$

Turning to division and assuming $l_1 > l_2$, we find

$$T_I(x_3 \leftarrow x_1/x_2,\ x_4 \leftarrow x_1 \bmod x_2) \sim l_2(l_1 - l_2), \tag{62}$$

since this involves essentially the same work as computing the product x_2x_3.

In studying the computation of $x_3 = \gcd(x_1, x_2)$ by Euclid's algorithm (Section 1.4), we again assume $l_1 > l_2$, and we view each division step as a sequence of subtraction steps. Since the total number of subtraction steps is bounded by $2(l_1 - l_3)$, and the work in each of these steps is dominated by l_2, we have

$$T_I(x_3 \leftarrow \gcd(x_1, x_2)) \sim l_2(l_1 - l_3). \tag{63}$$

The bound is achieved when the IRS is a Fibonacci sequence.

Finally we consider the Chinese remainder algorithm (CRA) for integers, using the notation of Section 4.8. Let l_1 and l_2 be the lengths of m_1 and m_2, respectively. By (63), the time to compute c in Step (1) is codominant with l_1l_2. Since Steps (2) and (3) can also be performed within this bound, we have

$$T_I(\text{CRA}) \sim l_1l_2. \tag{64}$$

5.4 POLYNOMIAL OPERATIONS. Let $T_P(\text{op})$ denote the maximum computing time for the polynomial operation op, and let F_i denote a polynomial in v variables with dimension vector (l_i, d_i), where $l_i > 0$. Clearly the number of terms in F_i is at most $(d_i + 1)^v$, and this bound is *not* codominant with d_i^v.

For addition and subtraction it is easy to show that

$$T_P(F_3 \leftarrow F_1 \pm F_2) \sim l_1(d_1 + 1)^v + l_2(d_2 + 1)^v, \tag{65}$$

while for classical multiplication

$$T_P(F_3 \leftarrow F_1F_2) \sim l_1l_2(d_1 + 1)^v(d_2 + 1)^v. \tag{66}$$

Turning to division, if $F_2 \mid F_1$, we find

$$T_P(F_3 \leftarrow F_1/F_2) \sim l_2l_3(d_2 + 1)^v(d_3 + 1)^v, \tag{67}$$

since this involves essentially the same work as computing the product F_2F_3.

If $F_2 \nmid F_1$, division yields to pseudo-division (Section 2.3), which is more expensive because of coefficient growth. In this case, suppose that F_1 and F_2 have degrees d_1 and d_2, respectively, in the main variable, and that their coefficient polynomials (in the $v - 1$ auxiliary variables) have dimension vectors bounded by (l, d). Let (l', d') bound the dimension vectors of the polynomial coefficients of the pseudo-quotient, Q, and note that the degree of Q in the main variable is $\delta = d_1 - d_2$. Since pseudo-division (PDIV) involves essentially the same work as computing the product QF_2, we have by (66)

$$T_P(\text{PDIV}) \sim (\delta + 1)(d_2 + 1)ll'(d + 1)^{v-1}(d' + 1)^{v-1}. \tag{68}$$

Now it can be shown that $d' \leq (\delta + 2)d$, and similarly [using assumption (A3)] $l' \leq (\delta + 2)l$. On the other hand, for large random problems we find that $d' > d$, $l' > l$, and $\delta = 1$. Hence

$$\begin{aligned} T_P(\text{PDIV}) &\lesssim l^2(d_2 + 1)(\delta + 2)^{v+1}(d + 1)^{2v-2}, \\ T_P(\text{PDIV}) &\gtrsim l^2(d_2 + 1)(d + 1)^{2v-2}. \end{aligned} \tag{69}$$

For polynomials in $Z_p[x]$, assumption (A4) permits us to restrict our attention to single-word primes; hence all coefficient operations can be performed in some fixed amount of time. For Algorithm U, let $d_1 = \partial(F_1)$ and $d_2 = \partial(F_2)$, and suppose $d_1 \geq d_2 > 0$. Then, by essentially the same argument that led to (63),

$$T_P(\text{U}) \sim d_2(d_1 - d_3), \tag{70}$$

where d_3 is the degree of the GCD. For the Chinese remainder algorithm, we again use the notation of Section 4.8. Let d_1 and d_2 be the degrees of m_1 and m_2, respectively. By (70), the time to compute c in Step (1) is codominant with d_1d_2. Since Steps (2) and (3) can also be performed within this bound, we have

$$T_P(\text{CRA}) \sim d_1d_2\,. \tag{71}$$

5.5 ALGORITHM C. In analyzing the computing time for Algorithm C, we shall use the notation of Section 2.4, and we shall assume that Step (4) is performed by means of the subresultant PRS algorithm (Section 3.6). Let F_1' and F_2' be the given nonzero polynomials in v variables, and let (l, d) bound their dimension vectors.

Let $C(v, l, d)$ denote the maximum computing time for Algorithm C, and let $C_i(v, l, d)$ denote the time for the ith step. We shall omit the analyses of several steps which obviously make no contribution to the final bound.

Step (1). Since this step involves at most $2d + 1$ GCD's of polynomial coefficients, we have

$$C_1(v, l, d) \lesssim d \cdot C(v - 1, l, d). \tag{72}$$

Step (2). By (67), the time required to divide c_1 or c_2 into a coefficient of F_1' or F_2' is dominated by $l^2(d + 1)^{2v-2}$. Since there are at most $2(d + 1)$ such divisions, we have

$$C_2 \lesssim l^2(d + 1)^{2v-1}. \tag{73}$$

Step (4). Clearly most of the work in this step is in the pseudo-divisions which yield $\text{prem}(F_{i-2}, F_{i-1})$ for $i = 3, \cdots, k$. As in Section 3.1, let d_i denote the degree of F_i in the main variable. Then by assumption (A1), we have $d_i = d_2 + 2 - i$, for $i = 3, \cdots, k$, and therefore $k \leq d_2 + 2$. Furthermore, by (20) the degree of F_i in any auxiliary variable cannot exceed $d(d_1 + d_2) \leq 2d^2$. Similarly, by (22), ignoring the logarithm terms in accordance with assumption (A3), the integer-length of F_i cannot exceed $l(d_1 + d_2) \leq 2ld$. Hence we can bound the time for a single pseudo-division by replacing l by $2ld$, d_2 by d, δ by 1, and d by $2d^2$ in (69). Multiplying the result by d, which bounds the number of pseudo-divisions, and applying some obvious simplifying inequalities, we obtain

$$C_4 \lesssim l^2(d + 1)^{4v}2^{2v}3^v. \tag{74}$$

Step (6). To bound the time for computing $f_k/\bar{g}$, we can replace v by $v - 1$, l_2 by l, d_2 by d, l_3 by $2ld$, and d_3 by $2d^2$ in (67); the result is dominated by $l^2(d + 1)^{3v-2}2^v$. To bound the time for dividing this quantity into a coefficient of F_k, we can replace v by $v - 1$, l_2 by $2ld$, d_2 by $2d^2$, l_3 by l, and d_3 by d in (67), with the same result. Multiplying by $d + 2$, which bounds the number of such divisions, we find

$$C_6 \precsim l^2(d + 1)^{3v-1}2^v. \tag{75}$$

Step (7). By assumption (A3), the dimension vectors for $\bar{g}$ and for the coefficients of G are bounded by (l, d); hence the dimension vectors for the coefficients of $\bar{G}$ and for $\operatorname{cont}(\bar{G})$ are bounded by $(2l, 2d)$. Now by the argument used in Step (1), $\operatorname{cont}(\bar{G})$ can be computed in time $d \cdot C(v - 1, 2l, 2d)$. Also, by the argument used in Step (2), each division of this content into a coefficient of $\bar{G}$ can be performed in time $l^2(d + 1)^{2v-2}2^v$. Hence,

$$C_7(v, l, d) \precsim d \cdot C(v - 1, 2l, 2d) + l^2(d + 1)^{2v-1}2^v. \tag{76}$$

Step (8). Here (l, d) bounds the dimension vectors of c and of the coefficients of G. By (66), the time to multiply one of these coefficients by c is at most ${}^2(d + 1)^{2v-2}$; hence

$$C_8 \precsim l^2(d + 1)^{2v-1}. \tag{77}$$

Total Time. Summing these bounds, we find

$$C(v, l, d) \precsim d \cdot C(v - 1, 2l, 2d) + l^2(d + 1)^{4v}2^{2v}3^v, \tag{78}$$

for $v > 0$. Starting with the formula

$$C(0, l) \precsim l^2, \tag{79}$$

which follows from (63), we can now prove by induction on v that

$$C(v, l, d) \precsim l^2(d + 1)^{4v}2^{2v^2}3^v. \tag{80}$$

5.6 ALGORITHM P. In analyzing the computing time for Algorithm P, we shall use the notation of Section 4.5. Let F_1' and F_2' be the given nonzero polynomials in $Z_p[x_1, \cdots, x_v]$, and let d bound the components of their degree vectors.

Let $P(v, d)$ denote the maximum computing time for Algorithm P, and let $P_i(v, d)$ denote the time for the ith step. We shall omit the analyses of several steps which obviously make no contribution to the final bound.

Step (1). Since F_1' and F_2' each have at most $(d + 1)^{v-1}$ terms, and since the time to compute a GCD in the coefficient domain $\mathfrak{g} = Z_p[x]$ by Algorithm U is at most d^2 [by (70)], we have

$$P_1 \precsim (d + 1)^{v+1}. \tag{81}$$

Step (2). By (67), the time required to divide c_1 or c_2 into a coefficient of F_1' or F_2' is dominated by $(d + 1)^2$. Since there are at most $2(d + 1)^{v-1}$ such divisions, we have

$$P_2 \precsim (d + 1)^{v+1}. \tag{82}$$

Step (7). Let φ denote either $\bar{g}$ or any coefficient of F_1 or F_2. Thus $\varphi \in Z_p[x_v]$, and $\partial_v(\varphi) \leq d$. Since the time to map φ into $\varphi \bmod (x_v - b) = \varphi(b) \in Z_p$, either by division or by Horner's rule [1, p. 423], is dominated by $d + 1$, the time to map $\bar{g}$ and all of the coefficients of F_1 and F_2 into Z_p is dominated by $(d + 1)^v$. Since this dominates the time required for the ensuing multiplications, we have

$$P_7 \precsim (d + 1)^v. \tag{83}$$

Step (8). Here we invoke Algorithm P recursively, and then multiply the resulting GCD and cofactors by $\tilde{g}$ and $\tilde{g}^{-1}$, respectively. Thus

$$P_8(v, d) \precsim P(v - 1, d) + (d + 1)^{v-1}. \tag{84}$$

Step (11). Here each of the coefficients of G^*, H_1^*, and H_2^* must be extended from degree $n - 2$ in x_v to degree $n - 1$. By (71), the required time for each such extension is codominant with n; hence

$$P_{11} \precsim n(d + 1)^{v-1}. \tag{85}$$

Step (14). Recall that $G^* = \bar{G} = (\bar{g}/g)G$. By the same reasoning as in Step (1), $\text{cont}(G^*)$ can be computed in time $(d + 1)^{v+1}$. By the same reasoning as in Step (2), the ensuing divisions of this content into G^*, and of g into H_1^* and H_2^*, can also be performed in time $(d + 1)^{v+1}$. Hence

$$P_{14} \precsim (d + 1)^{v+1}. \tag{86}$$

Step (15). By (66), each multiplication of a coefficient of G by c can be performed in time $(d + 1)^2$. Hence the time to compute G' is bounded by $(d + 1)^{v+1}$. Since the same reasoning holds for H_1' and H_2', we have

$$P_{15} \precsim (d + 1)^{v+1}. \tag{87}$$

Total Time. By assumption (A2) no unlucky b-values will occur. Hence if F_1 and F_2 are relatively prime, only one b-value will be needed; otherwise the required number is

$$\begin{aligned} \bar{n} &= \bar{\nu} + 1 \\ &= \max\,(\partial_v(\bar{F}_1), \partial_v(\bar{F}_2)) + 1 \\ &\le 2d + 1. \end{aligned} \tag{88}$$

Summing the preceding bounds, with Steps (6)–(12) weighted by (88), we obtain

$$P(v, d) \precsim d \cdot P(v - 1, d) + (d + 1)^{v+1}, \tag{89}$$

for $v > 1$. Starting with the formula

$$P(1, d) \precsim d^2, \tag{90}$$

which follows from (70), we can now prove by induction on v that

$$P(v, d) \precsim (d + 1)^{v+1}. \tag{91}$$

5.7 ALGORITHM M. In analyzing the computing time for Algorithm M, we shall use the notation of Section 4.3. Let F_1' and F_2' be the given nonzero polynomials in $Z[x_1, \cdots, x_v]$, and let (l, d) bound their dimension vectors.

Let $M(v, l, d)$ denote the maximum computing time for Algorithm M, and let $M_i(v, l, d)$ denote the time for the ith step. In view of the similarity of Algorithm M to Algorithm P, we shall merely present the results of the analyses of the significant steps:

$$
\begin{aligned}
M_1 &\lesssim l^2(d+1)^v, \\
M_2 &\lesssim l^2(d+1)^v, \\
M_7 &\lesssim l(d+1)^v, \\
M_8 &\lesssim P(v,d) + (d+1)^v \lesssim (d+1)^{v+1}, \\
M_{11} &\lesssim n(d+1)^v, \\
M_{14} &\lesssim l^2(d+1)^v, \\
M_{15} &\lesssim l^2(d+1)^v.
\end{aligned} \tag{92}
$$

Total Time. By assumption (A2) no unlucky primes will occur. Hence if F_1 and F_2 are relatively prime, only a single prime will be needed; otherwise the required number is

$$\bar{n} \leq \log_\alpha \mu^{**}, \tag{93}$$

where μ^{**} is the final value of μ^* in Step (13). Clearly μ^{**} need not exceed $2c^2t$, where t is the number of terms in $\bar{G}$, and where c bounds the magnitudes of the coefficients of $\bar{G}$, $\bar{H}_1$, and $\bar{H}_2$. It follows immediately that $\bar{n} \leq 2 \log_\alpha c + \log_\alpha t + \log_\alpha 2$. But by assumption (A3), $\log_\alpha c \leq 2l$, and by assumption (A4), $\log_\alpha t < 1$. Hence

$$\bar{n} \leq 4l + 2. \tag{94}$$

Summing (92) with Steps (6)–(12) weighted by (94), we obtain

$$M(v, l, d) \lesssim l^2(d+1)^v + l(d+1)^{v+1}. \tag{95}$$

When F_1 and F_2 are relatively prime (RP), it suffices to sum Steps (1)–(9) and (15); thus we find the smaller bound

$$M^{(\mathrm{RP})}(v, l, d) \lesssim l^2(d+1)^v + (d+1)^{v+1}. \tag{96}$$

5.8 COMPARISON. From (80) and (95), we see that the bound on M is strictly dominated by the bound on C. We shall now prove that M is strictly dominated by C in the region $v \geq 2$.

Let F_1 and F_2 be random polynomials in v variables subject only to the constraints imposed by the dimension vector (l, d), and let C_- be the maximum computing time to obtain their pseudo-remainder F_3. Then by (69), C_- dominates $l^2(d+1)^{2v-1}$. On the other hand, by (95), M is strictly dominated by $M_+ = l^2(d+1)^{v+1}$. Since $C_-/M_+ = (d+1)^{v-2}$, it follows that $M < M_+ < C_- \lesssim C$ in the region $v \geq 2$. This completes the proof.

6. *Summary and conclusions*

In attempting to generalize Euclid's algorithm to the case of univariate or multivariate polynomials with integer or rational coefficients, one immediately encounters the problem of coefficient growth. This is most serious when the growth is allowed to go unchecked, as in the Euclidean PRS algorithm, or when the algorithm is recursively applied to inflated polynomial coefficients, as in the multivariate case of the primitive PRS algorithm.

Since the subresultant PRS algorithm restricts the coefficient growth to a linear rate comparable to that which often occurs in a primitive PRS, and at the same time avoids the need for recursive applications to inflated coefficients, it appears to satisfy the most optimistic criteria that could be set for Algorithm C. Nevertheless, it may require the production of subresultants much larger than either the given polynomials or their GCD.

Algorithm M avoids this difficulty by taking a fundamentally different approach. The given polynomials are first projected by modular mappings into one or more simpler domains in which images of the GCD can more easily be computed. The true GCD is then constructed from these images with the aid of the Chinese remainder algorithm. Since the same method is used for the required GCD computations in the image spaces, it is only necessary to apply Euclid's algorithm to integers and to univariate polynomials with coefficients in a finite field.

This modular approach is especially well suited to the problem of GCD computation, because the desired GCD is typically smaller than the given polynomials, and this limits the required number of images. In particular when the GCD is unity, a single image is usually sufficient.

Of critical importance to both Algorithm C and Algorithm M is the existence of a small multiple of the GCD whose leading coefficient can be computed with negligible effort. Otherwise, it would be necessary in Algorithm M to obtain enough images to build the associated subresultant, whose coefficients might be very large, and it would be necessary in both algorithms to compute the primitive part of that subresultant.

A striking difference between the two algorithms is that Algorithm S views a polynomial in $Z[x_1, \cdots, x_v]$ as a univariate polynomial in some main variable, with polynomial coefficients, while Algorithm M treats it directly as a multivariate polynomial with integer coefficients. It is easy to see that Algorithm M profits greatly from the resulting speed of operations on the larger number of smaller coefficients. This same idea is applied in the image space $Z_p[x_1, \cdots, x_v]$, whose polynomials are viewed by Algorithm P as polynomials in the variables $x_1, \cdots, x_{v-1}$ with coefficients in $Z_p[x_v]$.

In our study of computing times, we obtained asymptotic bounds on the maximum computing times for the two algorithms, with the aid of several simplifying assumptions which we believe to be realistic. Furthermore we showed that the maximum computing time for Algorithm M is strictly dominated by the maximum computing time for the first pseudo-division in Algorithm C, in the region $v \geq 2$. This dramatically illustrates the superiority of the modular approach for GCD computations in which the given polynomials are sufficiently large and sufficiently dense.

ACKNOWLEDGMENTS. The author is indebted to a number of colleagues including especially G. E. Collins and S. C. Johnson for comments on earlier drafts of this paper, and for helpful discussions.

REFERENCES

1. KNUTH, D. E. *The Art of Computer Programming, Vol. 2: Seminumerical Algorithms.* Addison-Wesley, Reading, Mass., 1969.
2. SAMMET J. E., ET AL. Symbol manipulation. *Comm. ACM 9* (Aug. 66), 547–643.
3. HOUSEHOLDER, A. S. Bigradients and the problem of Routh and Hurwitz, *SIAM Rev. 10* (Jan. 1968), 56–66.
4. HOUSEHOLDER, A. S., AND STEWART, G. W., III. Bigradients, Hankel determinants, and

the Padé table. In *Constructive Aspects of the Fundamental Theorem of Algebra*, B. Dejon and P. Henrici, Eds. Wiley-Interscience, New York, 1969, pp. 131–150.
5. USPENSKY, J. V. *Theory of Equations*, McGraw-Hill, New York, 1948.
6. BIRKHOFF G., AND MACLANE, S. *A Survey of Modern Algebra*, 3rd ed. Macmillan, New York, 1965.
7. COLLINS, G. E. Subresultants and reduced polynomial remainder sequences. *J. ACM 14* (Jan. 1967), 128–142.
8. BROWN W. S., AND TRAUB, J. F. On Euclid's algorithm and the theory of subresultants. *J. ACM 18* (Oct. 1971), 505–514.
9. GOLDSTEIN A. J., AND GRAHAM, R. L. A Hadamard type bound on the coefficients of a determinant of analytic functions (to be published).
10. COLLINS, G. E. The calculation of multivariate polynomial resultants. In *Proc. 2nd Symposium on Symbolic and Algebraic Manipulation*. ACM, New York, 1971, pp. 212–222; *J. ACM 18* (Oct. 1971), 515–532.
11. ERDÉLYI, A. *Asymptotic Expansions*. Dover, New York, 1956.
12. *Proc. 2nd Symposium on Symbolic and Algebraic Manipulation*. ACM, New York, 1971, Chs. 3 and 7.