On Implementing Buchberger's Algorithm for Gröbner Bases

S.R. Czapor[1]

K.O. Geddes[2]

[1] *Department of Applied Mathematics*
[2] *Department of Computer Science*
University of Waterloo
Waterloo, Ontario
Canada N2L 3G1

ABSTRACT

An implementation in the Maple system of Buchberger's algorithm for computing Gröbner bases is described. The efficiency of the algorithm is significantly affected by choices of polynomial representations, by the use of criteria, and by the type of coefficient arithmetic used for polynomial reductions. The improvement possible through a slightly modified application of the criteria is demonstrated by presenting time and space statistics for some sample problems. A fraction-free method for polynomial reduction is presented. Timings on problems with integer and polynomial coefficients show that a fraction-free approach is recommended.

1. Introduction

The method of Gröbner bases provides an important technique for (among other things) solving the simplification problem over polynomial ideals, and solution of algebraic equations. Hence, an implementation of Buchberger's algorithm would seem to be an almost essential feature of any advanced algebra system. It came as a surprise, while working on an implementation for the Maple system ([10], [11]), just how large a Gröbner basis problem can result from simple input polynomials given an apparently careful implementation of the improved algorithm, as described in [5]. This is particularly true when the lexicographic ordering of terms is used, as the algorithm is known to be sensitive to permutations of the variable ordering in this case. Nonetheless, lexicographic ordering is of some practical importance for solving algebraic systems. It provides a natural elimination and separation of variables in a manner compatible with resultants (see [17]), and can result in an elegant form of reduced system (see Example 6.15 in [5]) with which back-solving is simple. Moreover, it applies even when infinitely many solutions exist (cf. Method 6.12 in [5]).

This research was supported by the Natural Sciences and Engineering Research Council of Canada under Grant A8967.

Using empirical observations, we found a number of modest improvements which cumulatively allow run-time to be reduced significantly (up to a factor of five). These might also be applied when the total degree ordering is used; however, as this was not the primary focus of our investigations, we give results only for lexicographic problems.

We assume the basic notation of Buchberger ([2], [3], [5]), and for brevity omit basic definitions wherever possible. In the next section we specify the variant of Buchberger's algorithm chosen, and briefly describe its implementation. This includes a slight variant of the typical use of the criteria for avoiding unnecessary reductions, which often eliminates a good deal more unnecessary computation. In section 3 we examine a number of different approaches to polynomial reduction, and briefly assess their relative performance.

2. Some details on the implementation

Since several variants of the basic algorithm are possible, we will first outline the one used in the Maple implementation. Let $F = \{f_1, \ldots, f_k\}$ be a set of polynomials (over some field) in the indeterminates $X = \{x_1, \ldots, x_n\}$, and let $<_T$ be a total ordering of X-terms. Then the following algorithm produces the reduced, minimal Gröbner basis for F.

Algorithm 2.1

$G \leftarrow Minor(F)$; $k \leftarrow length(G)$;
$B \leftarrow \{[i,j] \mid 1 \le i < j \le k$ and $Criterion2([i,j], G)\}$
while $B \ne \emptyset$ **do**
  $B1 \leftarrow \{[i,j] \mid [i,j] = SelN(B,G)$ and
    $\neg\, Criterion1([i,j], B, G)\}$
  while $B1 \ne \emptyset$ **do**
    $B \leftarrow B - B1$; $B1 \leftarrow \{$ (*as above*) $\}$
  $[i,j] \leftarrow SelN(B,G)$; $B \leftarrow B - \{[i,j]\}$
  $f \leftarrow NormalF(Spoly(G_i, G_j), G)$
  if $f \ne 0$ **then**
    $G \leftarrow G \cup \{f\}$; $k \leftarrow k+1$
    $B \leftarrow B \cup \{[i,k] \mid 1 \le i < k$ and $Criterion2([i,k], G)\}$
$G \leftarrow Minor(G)$

The procedure *Minor* consists of the procedures *ReduceAll* and *NewBasis* given in [5]. It produces a set in which each polynomial is in "normal form" w.r.t. the other polynomials. The procedures *Criterion1* and *Criterion2* are the following criteria ([4], [6]) for detecting unnecessary S-polynomial reductions:

$$Criterion1([i,j], B,G) \equiv \neg \exists\, u,\, 1 \le u \le k \;\; s.t. \qquad (2.1a)$$
$$\{\; i{\ne}u{\ne}j,\; [i,u] \notin B,\; [u,j] \notin B,\; H_G(u) \mid H_G(i,j) \;\},$$

$$Criterion2([i,j], G) \equiv H_G(i)\, H_G(j) \ne H_G(i,j). \qquad (2.1b)$$

Finally, *SelN* is a procedure which selects a pair from the set B according to the "normal selection strategy":

$$SelN(B,G) \equiv [i,j] \in B \;\; such\ that \qquad (2.2)$$
$$H_G(i,j) = \min_{<_T} \{H_G(u,v) \mid [u,v] \in B\},$$

where

$$H_G(i,\, j) = lcm(hterm(G_i),\, hterm(G_j)). \qquad (2.3)$$

Algorithm 2.1 differs from the "standard" formulation in two important respects. First, the procedure *Criterion2* is applied *before* adding new pairs to the set B (as opposed to after selection of [i,j] $\in$ B). Using the lexicographic ordering, many pairs will fail to satisfy *Criterion2*. Hence, significant savings in overhead may result from avoiding them altogether. (Note that *Criterion2* does not depend on B.) The second difference stems from the fact that *SelN* is, in fact, many-valued. Using the lexicographic ordering, there are often many pairs which satisfy *SelN* at a given point. It happens that some of these may satisfy *Criterion1* (which depends on B), while others may not. Hence, we may pass through the inner loop of Algorithm 2.1 several times, until *all* pairs given by *SelN* satisfy our criterion. In this way, we might detect more unnecessary reductions than if we had considered the pairs one at a time.

We now illustrate the sort of improvement (for our implementation) that these simple modifications can yield. The timings were done on a Vax 11/785 processor running Maple version 4.0, and using a fraction-free reduction arithmetic described in the next section (see "prim3"). The test problems are described in the Appendix.

Problem		Algorithm Type	
		standard	(2.1)
1	unnecessary reductions	6	2
	time (sec)	106	58
	storage (K bytes)	1103	819
3	unnecessary reductions	74	35
	time	1070	553
	storage	1262	1221
5(a)	unnecessary reductions	83	55
	time	4918	2346
	storage	1540	1491
4(b)	unnecessary reductions	152	110
	time	71 259	37 295
	storage	2851	2736

Currently, two coefficient domains are handled by the Maple implementation, namely the rational numbers (Q), and rational functions (over Q). In the former domain, all polynomials are kept in distributed (expanded) form, and are hence manipulated in the same mathematical format in which they are input. This makes some processes (such as computing head-terms) slightly cumbersome, but allows all arithmetic to be done by system functions; it is therefore relatively fast and simple. This approach is not feasible for the latter coefficient domain, primarily because of space considerations. In this case, polynomials are represented in "partially distributed" form. That is, each term over X appears only once, and with a fully distributed coefficient. To facilitate this, we represent a polynomial as a sparse table whose indices are the terms of the polynomial and whose entries are the corresponding coefficients. As a result, much larger problems can be handled without exceeding space limitations.

Also, there is a choice of either the graduated (total degree) or the lexicographic term ordering. If a list of indeterminates $X = [x_1, \cdots, x_n]$ is specified, the order is based on $x_1 > \cdots > x_n$; if a set $\{x_1, \cdots, x_n\}$ is given, it will be permuted according to the heuristic in [1].

3. Polynomial reduction: a fraction-free approach

In [17], Pohst and Yun exploited the relationship between Buchberger's algorithm (using lexicographic ordering) and polynomial remainder sequences. Roughly, one can view an S-polynomial or reduction of one polynomial modulo another as a generalized division step. In turn one then views a pseudo-remainder as a series of division steps, and a resultant as a series of pseudo-remainders. The standard approach to polynomial reduction seems to be to work over the fraction field Q_I of the base [integral] domain I (e.g. Q when $\mathbb{Z}$ is the base domain), in a manner similar to the Monic Euclidean PRS Algorithm for GCD computation ([15]). Noting the superiority of the Primitive PRS Algorithm in that context (see [14]), we will examine the possibility of avoiding the fraction field in a similar manner for the reduction process.

By definition, a polynomial p is in *normal form* w.r.t. a set of polynomials $F = \{F_1, F_2, \ldots, F_k\}$ iff no headterm of any polynomial in F divides any term of p. If instead $p = \alpha t + q$, and $\exists\, F_j, p, u$ such that

$$hterm\,(F_j) \mid t \;\;,\;\; \alpha{\cdot}t = \beta{\cdot}u{\cdot}\, head\,(F_j), \qquad (3.1)$$

then p *reduces w.r.t.* F_j. For efficiency, we reduce those terms which are $<_T$-maximal first, noting that p is in normal form (mod F) iff

(a) $hterm(p)$ is irreducible, (3.2a)

(b) $rest(p)$ is in normal form $(mod F)$. (3.2b)

The simplest approach to the arithmetic in (3.1) is to choose

$$\beta = \frac{\alpha}{hcoeff(F_j)} \qquad (3.3)$$

so that

$$p > (\alpha t + q) - \beta{\cdot}u{\cdot}F_j = p'. \qquad (3.4)$$

This is more efficient if the F_j have been divided by their respective head coefficients (i.e., made monic over X). We might carry this a step further, re-scaling the reduced polynomial after every reduction, in the hope of controlling coefficient growth a little better. This involves more work if p' is larger than any polynomial in F, as will often be the case.

To avoid rationals altogether, we have to cross-multiply. If $\nu = hcoeff(F_j)$, we compute $\gamma = gcd(\nu, \alpha)$ and

$$p \gg (\frac{\nu}{\gamma})(\alpha t + q) - (\frac{\alpha}{\gamma}) \cdot u \cdot F_j . \qquad (3.5)$$

In the Primitive PRS algorithm, it is obvious that one should remove the content of *each* pseudo-remainder; otherwise, it will be carried into the next step of the sequence, producing rapid expression swell. On the other hand, if our reducing basis F contains only primitive polynomials, there is no reason to expect growth of the same magnitude during reduction of another polynomial. It may happen that extraction of the primitive part is only essential just before appending the reduced polynomial to F (and using it in subsequent reductions). If this were the case, the overhead of the content sub-computations would not be prohibitive. Also, it is often possible to compute the content of the reduced polynomial very efficiently, using a probabilistic algorithm ([16]). It is natural to compare three approaches:

(prim1) - remove content only when normal form is obtained,

(prim2) - remove content when the coefficients become "large", and when normal form is obtained,

(prim3) - remove content after each single reduction,

where, for example, the "prim2" scheme might be arranged as follows.

```
procedure Normalf(f,G)
    f ← Hreduce(f, G, 1)
    h ← head(f) ; k ← h ; rest ← f - h
    contin ← 1
    while rest ≠ 0 do
        temp ← Hreduce(rest,G,contin,scale,contout)
        if temp ≠ rest and contin ≠ contout then
            ck ← content(k)
            k ← k/ck ; scale ← ck·scale
            fcont ← gcd(scale, contout)
            contout ← contout/fcont
            scale ← scale/fcont
        h ← head(temp) ; rest ← temp - h
        k ← scale·k + contout·h
        contin ← contout
    return( k/content(k) )
```

```
procedure Hreduce(f,G, contin, scale, contout)
    big ← 3·max{ length(hcoeff(G_i)) | G_i ∈ G}
    ascale ← 1 ; acont ← contin ; n ← |G|
    for j from 1 to n while f ≠ 0 do
        if hterm(G_j) | hterm(f) then
            u ← hterm(f)/hterm(G_j)
            temp ← gcd(hcoeff(G_j), hcoeff(f) )
            m1 ← hcoeff(G_j)/temp
            m2 ← hcoeff(f)/temp
            f ← (m1·f) - (m2·u·G_j)
            ascale ← m1·ascale
            if f ≠ 0
            and length(hcoeff(f)) > big then
                temp ← content(f) ; f ← f/temp
                acont ← acont·temp
            j ← 0
    scale ← ascale; contout ← acont
    return(f)
```

Needless to say, there are other (perhaps better) ways to compute the bound "big" in the above. We did not attempt to fine-tune this calculation.

Since different system functions are involved for the two domains considered, we first examine problems with integer coefficients, comparing the three previous codes with the following:

(monic1) - rescale (i.e. make monic) after each reduction,

(monic2) - rescale when normal form is obtained.

All times are in seconds; the space statistics (in K bytes) are parenthesized.

Problem	Reduction Type				
	monic1	monic2	prim1	prim2	prim3
3	584 (1188)	570 (1172)	560 (1200)	555 (1221)	553 (1221)
5(a)	3125 (1548)	2833 (1564)	2366 (1500)	2348 (1500)	2346 (1491)
4(a)	4882 (1204)	3617 (1220)	2294 (1753)	2209 (1442)	2380 (1344)
2	235 (812)	223 (820)	97 (1106)	103 (1090)	103 (1057)
1	143 (762)	136 (754)	55 (836)	58 (828)	58 (819)
4(b)	109675 (2752)	90759 (2802)	40427 (4891)	38420 (2851)	37295 (2736)

We first note that the extra re-scaling done in the "monic1" scheme does not seem to yield any significant savings in space. Next we note that the gap between the analogous schemes "monic2" and "prim3" widens as the size of the coefficients increases. Finally, it requires a large problem to confirm that "prim3" is indeed *asymptotically* better than "prim1". (The potential gain of the less conservative schemes is likely small because of the relative speed of Maple's integer *gcd* and *content* functions; unlike their polynomial counterparts, these functions are part of Maple's compiled kernel.)

When there are free parameters present, we proceed more cautiously because of the increased cost/complexity of the GCD subcomputations involved. First, we note that the content can often be computed with a single GCD computation, as follows:

procedure *Gcontent*(p)
  ic $\leftarrow$ *integer_content*(p) ; p $\leftarrow$ p/ic
  h $\leftarrow$ *head*(p) ; ch $\leftarrow$ *hcoeff*(p)
  k $\leftarrow$ a monomial in (p-h) ; ck $\leftarrow$ *hcoeff*(k)
  g1 $\leftarrow$ *gcd*(ch, ck) ; g1 $\leftarrow$ g1/*integer_content*(g1)
  if g1 | p **then return** (ic $\cdot$ g1)
  else return ic $\cdot$*content*(p - h - k+ $g1\ x_1^{degree(p,x_1)+1}$)

Since the head coefficient is often smallest, the GCD of the coefficients of p is often $g1$ (up to an integer multiple). Otherwise, we impose a dummy head coefficient (term) on p, neglect the coefficients (terms) already considered, and compute the content by another method (such as Maple's standard *content* function, or by repeating *Gcontent*).

Second, we expect the effects of coefficient growth to be more pronounced. Ideally, we would like to control this growth as much as possible without having to compute the content after each reduction. Since no counterpart to Collins' Reduced PRS algorithm is available, we consider instead a trial division approach similar to that of Hearn ([13]) for the Primitive PRS. While Hearn's algorithm uses only the leading coefficients of the sequence polynomials as trial divisors, it is necessary in the present case to look for *ratios* of head coefficients. Fortunately, (for lexicographic ordering) these coefficients typically contain many of the same factors; hence their irreducible components can often be obtained by trial divisions alone, as follows. If

$$H = \{hcoeff(F_1), hcoeff(F_2), ..., hcoeff(F_k)\},$$

then we use as trial divisors $D = cdecomp(H)$, where *cdecomp* is defined as:

procedure *cdecomp*(H)
  $D \leftarrow \emptyset$; $R \leftarrow H - (H \cap \mathbb{Z})$
  while $R \neq \emptyset$ **do**
    $k \in R$; $R \leftarrow R - \{k\}$
    for $d \in D$ **do while** $d \mid k$ **do** $k \leftarrow \frac{k}{d}$
    if $k \notin \mathbb{Z}$ **then**
      $D_o \leftarrow \emptyset$
      for $d \in D$ **do**
        if $k \mid d$ **then**
          $D_o \leftarrow D_o \cup \{d\}$
          $R \leftarrow R \cup \{\frac{d}{k}\}$
      $D \leftarrow D - D_o \cup \{k\}$
      $R \leftarrow R - (R \cap \mathbb{Z})$
  return D

This appears to work well, in that the components obtained are often irreducible without further (formal) factoring, and length(D) is usually less than length(H). When a new polynomial (with head coefficient h) is added to the basis, the set of trial divisors becomes $cdecomp(D \cup \{h\})$. Several combinations with the previous schemes are possible; we implemented the following:

(prim4) - after each reduction, perform trial divisions; remove any left-over content via *Gcontent* at the end of each *Hreduce* call,

which compares most directly to the "prim3" scheme.

The various schemes compare as follows.

Prob.	Reduction Type					
	monic1	monic2	prim1	prim2	prim3	prim4
5(b)	26195 (2032)	21635 (2032)	4005 (2336)	5023 (2245)	5686 (2286)	5960 (2196)
8(b)	54971 (2441)	46875 (2597)	9511 (2851)	10081 (2810)	10744 (2622)	11921 (2425)
8(a)	225 (1088)	200 (1073)	113 (1073)	117 (1048)	142 (1134)	143 (1114)
6	1111 (1319)	832 (1188)	197 (1240)	250 (1204)	303 (1204)	241 (1261)
7	602 (1458)	230 1124	40 (795)	66 (959)	225 (1409)	55 (795)
9(a)	505 (1221)	375 (1148)	735 (1484)	969 (1476)	1014 (1468)	219 (1220)
8(c)	20184 (2613)	11759 (2220)	5039 (2818)	6928 (2744)	7482 (2810)	4785 (2441)
9(b)	(i) (ii)	(i) (ii)	(i) (iii)	(i) (ii)	(i) (ii)	58850 (10132)

(i) >125000 sec. ; (ii) >13000 K bytes ; (iii) >16000 K bytes

It should be noted that Maple's GCD polyalgorithm handles most simple GCD problems very efficiently without Hensel techniques (see [9] for a description of "*gcdheu*"). With this in mind, several conclusions are suggested by the above results. First, the "monic" schemes can be *slightly* more space-efficient, but *much* slower. (The sole exception is Problem 9(a), in which the GCD sub-problems encountered in the monic schemes are done via *gcdheu*; note, however, that the "prim4" scheme is still significantly faster.) In fact, because of the efficiency of *gcdheu* and *Gcontent*, the trial division code "prim4" is actually slower on one-parameter problems. It is, however, remarkably effective on multi-parameter problems. A more efficient combination of this technique with a primitive reduction scheme might involve "prim2", suitably tuned.

Acknowledgements

We are deeply indebted to Michael Monagan, who suggested the idea of avoiding the fraction field to us. We have also benefitted from many helpful discussions involving the Maple system, particularly with regard to implementation of the "partially distributed" polynomial representation.

Appendix: List of test problems

It should be noted that the variable orderings used are not necessarily optimal, either in the strict sense or that of [1]. In many cases we have deliberately permuted the variables to produce more difficult test problems.

Problem 1 : Trinks system [19], using the variable ordering $w>p>z>t>s>b$. Using Algorithm 2.1, 11 new polynomials are created. If we work over the integers, the largest head coefficient which appears is 10^{84}.

Problem 2 :

$$F_1 = 8x^2-2xy-6xz+3x+3y^2-7yz+10y+10z^2-8z-4,$$

$$F_2 = 10x^2-2xy+6xz-6x+9y^2-yz-4y-2z^2+5z-9,$$

$$F_3 = 5x^2+8xy+4xz+8x+9y^2-6yz+2y-z^2-7z+5,$$

using $x > y > z$.
(17 new polynomials are created; the largest head coefficient is 10^{74}.)

Problem 3 : Butcher's system (see [8]) of 8 equations in 8 unknowns, using the ordering $b_1 > a_{32} > b_2 > b_3 > a > c_3 > c_2 > b$.
(36 new polynomials are created; the largest head coefficient is 10^5.)

Problem 4 : Katsura's system (see [1]) of 5 equations in 5 unknowns.

(a) Use the ordering $u_4 > u_0 > u_2 > u_3 > u_1$.
(53 new polynomials are created; the largest head coefficient is 10^{59})

(b) Use the ordering $u_4 > u_0 > u_3 > u_2 > u_1$.
(140 new polynomials are created; the largest head coefficient is 10^{160})

Problem 5 ([12]):

$$F_1 = 2(b-1)^2+2(q-pq+p^2)+c^2(q-1)^2-2bq+2cd(1-q)(q-p)$$
$$+2bpqd(d-c)+b^2d^2(1-2p)+2bd^2(p-q)+2bdc(p-1)$$
$$+2bpq(c+1)+(b^2-2b)p^2d^2+2b^2p^2+4b(1-b)p+d^2(p-q)^2,$$

$$F_2 = d(2p+1)(q-p)+c(p+2)(1-q)+b(b-2)d+b(1-2b)pd$$
$$+bc(q+p-pq-1)+b(b+1)p^2d,$$

$$F_3 = -b^2(p-1)^2+2p(p-q)-2(q-1),$$

$$F_4 = b^2+4(p-q^2)+3c^2(q-1)^2-3d^2(p-q)^2$$
$$+3b^2d^2(p-1)^2+b^2p(p-2)+6bdc(p+q+pq-1).$$

(a) Substitute $b=2$, and use the ordering $q>c>p>d$.
(76 new polynomials are created; the largest head coefficient is 10^{29}.)

(b) Consider b a free parameter, and use the ordering $q>c>d>p$.
(24 new polynomials are created; the largest head coefficient is of degree 24 in b.)

Problem 6: Butcher's system (see [7]) of 3 equations in 3 unknowns, with 2 free parameters, using the variable ordering $a > b > g$.
(10 new polynomials are created; the largest head coefficient is of total degree 18.)

Problem 7 :

$$F_1 = ax^2 + bxy + cx + dy^2 + ey + f,$$

$$F_2 = bx^2 + 4dxy + 2ex + gy^2 + hy + k,$$

using $x > y$.
(2 new polynomials are created; the largest head coefficient is of total degree 6.)

Problem 8: Rimey's system [18] of 3 equations in 3 unknowns, with 4 free parameters, with the ordering of variables $x > y > z$.

(a) Substitute $\alpha = \beta = 1$.
(14 new polynomials are created; the largest head coefficient is of total degree 4.)

(b) Substitute $\epsilon = \lambda = \beta = 1$ in all polynomials, and $\alpha = 1$ in f only.
(48 new polynomials are created; the largest head coefficient is of degree 27 in α.)

(c) Substitute $\beta = 1$ in all polynomials, and $\alpha = 1$ in f, h.
(40 new polynomials are created; the largest head coefficient is of total degree 12.)

Problem 9 :

$$F_1 = x^2 + ayz + dx + g,$$

$$F_2 = y^2 + bzx + ey + h,$$

$$F_3 = z^2 + cxy + fz + k,$$

using $x > y > z$.

(a) Substitute d = e = f = 0.
(13 new polynomials are created; the largest head coefficient is of total degree 12.)

(b) Substitute a = b = c = g = h = k = 1.
(15 new polynomials are created; the largest head coefficient is of total degree 12, and dense.)

References

[1] W. Böge, R. Gebauer, H. Kredel: "Some Examples for Solving Systems of Algebraic Equations by Calculating Gröbner Bases", J. Symbolic Computation, Vol. 1, 1985 (to appear).

[2] B. Buchberger: "A Theoretical Basis for the Reduction of Polynomials to Canonical Forms", ACM SIGSAM Bull., Vol. 10, No. 3, Aug. 1976.

[3] B. Buchberger: "Some Properties of Gröbner-Bases for Polynomial Ideals", ACM SIGSAM Bull., Vol. 10, No. 4, Nov. 1976.

[4] B. Buchberger: "A criterion for detecting unnecessary reductions in the construction of Gröbner bases", Proc. EUROSAM '79, Marseille, June 1979, (W. Ng, ed.), *Lecture Notes in Computer Science*, Vol. 72, 1979, pp.3-21.

[5] B. Buchberger: "Gröbner Bases: An Algorithmic Method in Polynomial Ideal Theory", in *Progress, directions and open problems in multidimensional systems theory*, (N.K. Bose, ed.), D. Reidel Publishing Co., 1985, pp.184-232.

[6] B. Buchberger, F. Winkler: "Miscellaneous Results on the construction of Gröbner bases for polynomial ideals", Tech. Rep. 137, Univ. of Linz, Math. Inst., 1979.

[7] J.C. Butcher: "Stability Properties for a General Class of Methods for Ordinary Differential Equations", SIAM J. Numer. Anal. 18, No. 1, 1981, pp.37-44.

[8] J.C. Butcher: "An application of the Runge Kutta space", BIT Computer Science Numer. Math., Vol. 24, 1984, pp.425-440.

[9] B.W. Char, K.O. Geddes, G.H. Gonnet: "GCDHEU: Heuristic Polynomial GCD Algorithm Based on Integer GCD Computation", Proc. EUROSAM 84, Cambridge, July 1984, (J.P. Fitch, ed.), *Lecture Notes in Computer Science*, Vol. 174, 1984, pp.285-296.

[10] B.W. Char, K.O. Geddes, W.M. Gentleman, G.H. Gonnet: "The design of Maple: A compact, portable, and powerful computer algebra system", Proc. EUROCAL '83, *Lecture Notes in Computer Science*, Vol. 162, 1983, pp.101-115.

[11] B.W. Char, K.O. Geddes, G.H. Gonnet, S.M. Watt: "Maple User's Guide", WATCOM Publications, Waterloo, Ontario, 1985.

[12] G. Fee (Private communication.)

[13] A.C. Hearn: "Non-modular Computation of Polynomial GCD's using Trial Divisions", Proc. EUROSAM 79, Marseille, June 1979, (W. Ng, ed.), *Lecture Notes in Computer Science*, Vol. 72, 1979, pp.227-239.

[14] D. E. Knuth: *The Art of Computer Programming, Vol.2*, Addison-Wesley, Reading, Mass., 1969.

[15] R. Loos: "Generalized polynomial remainder sequences", in *Computer Algebra - Symbolic and Algebraic Computation*, (B. Buchberger, R. Loos and G.E. Collins, ed.), Springer, Wien New York, 2nd edition, 1983, pp.115-138.

[16] M.B. Monagan (Private communication.)

[17] M.E. Pohst, D.Y.Y. Yun: "On Solving Systems of Algebraic Equations via Ideal Bases and Elimination Theory", Proc. of the 1981 ACM Symposium on Symbolic and Algebraic Computation (SYMSAC '81), (P.S. Wang, ed.), Utah, Aug. 1981, pp.206-211.

[18] K. Rimey: "A System of Polynomial Equations and a Solution by an Unusual Method", ACM SIGSAM Bull., Vol. 18, No. 1, Feb. 1984.

[19] W. Trinks: "Über B. Buchbergers Verfahren, Systeme algebraischer Gleichungen zu losen", J. Number Theory, Vol. 10, No. 4, Nov. 1978, pp.475-488.