024

EUROCAL 87

Solving Algebraic Equations via Buchberger's Algorithm

S.R. Czapor

Department of Applied Mathematics
University of Waterloo
Waterloo, Ontario
Canada N2L 3G1

ABSTRACT

It is demonstrated that, when using Buchberger's Algorithm with the purely lexicographic ordering of terms, it is not generally feasible to inter-reduce basis polynomials during the progress of the algorithm. A heuristic is obtained (for polynomials over the rationals) which improves the efficiency of the reduction sub-algorithm, when the basis is not inter-reduced. Some improvements are made to a recent scheme for combining Buchberger's Algorithm with multivariate factorization. We present a hybrid variant of this scheme, in which extraneous sub-problems are detected outside of the lexicographic/elimination algorithm. Through this approach, the reduced solution bases for dense systems (previously impossible in the lexicographic ordering) may be found.

1. Introduction

For a system of algebraic equations having finitely many solutions, the Gröbner basis with respect to the graduated (or *total degree*) ordering of terms can be used to construct a set of univariate polynomials which give a finite inclusion of these solutions. This is done (when the structure of the basis indicates that there do exist only finitely many solutions) *without* an explicit elimination process. (See [3] for the details.) Hence, this procedure is often preferable to the triangularization provided by the lexicographical ordering of terms, which is sensitive to permutations of the variable ordering and can produce much more complex calculations.

Nonetheless, it is clear that this latter method cannot be ignored, since not all systems of interest have finitely many solutions. Moreover, there exist systems (e.g. those of Trinks, Hairer, Butcher found in [1]) for which the Gröbner basis is actually *easier* to compute with the lexicographic ordering (and a suitably chosen ordering of variables) because of special structure. Finally, there are systems (e.g. that of Rose, in [1]) for which computing the univariate polynomials [from the total degree basis] may be difficult because of the structure of the solution manifold - even though the basis itself is relatively easy to compute.

In this paper we discuss ways to improve the viability of using Buchberger's Algorithm with the lexicographic ordering of terms to solve algebraic equations. Consider a set F of polynomials in $K[x_1, ..., x_n]$, where K is a field and $x_1 > ... > x_n$, and let p, p_1, ... be any other such polynomials . We denote the leading monomial of p with respect to the lexicographic ordering by $M(p)$. We then denote

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

the corresponding coefficient and term by $hcoeff(p)$ and $hterm(p)$, respectively, and define

$$M(p_1, ..., p_m) \equiv lcm(M(p_1), ..., M(p_m)) , \tag{1.1}$$

$$Spoly(p_1, p_2) \equiv M(p_1, p_2) [\frac{p_1}{M(p_1)} - \frac{p_2}{M(p_2)}] . \tag{1.2}$$

Finally, we say that p is in *normal form modulo* F if no headterm of any polynomial in F divides any monomial in p. Then the simplest variant of Buchberger's Algorithm (including the improvements of [2]) to compute G, the Gröbner basis of F, is as follows.

Algorithm 1 ([2]):

G ← F ; k ← *length*(F) ; B ← {[i,j] | 1≤i<j≤k} ;
while B ≠ ∅ **do**
[i,j] ← *Select_Pair*(B, G) ; B ← B - {[i,j]}
if *Criteria*([i,j], B, G) **then**
h ← *NormalForm*(*Spoly*(G_i, G_j), G)
if h ≠ 0 **then**
(cutpoint 1)
k ← k+1 ; G ← G ∪ {h} ; B ← B ∪ {[i,k] | 1≤i<k }
return(*Fully_Reduce*(G))

The pair-selection procedure and criteria for avoiding zero-reductions are descibed in [3]. Also described is the procedure *Fully_Reduce*, which transforms the basis so that each polynomial is fully reduced modulo the others. This procedure could be invoked after *each* new polynomial is created; this results in another variant, which we refer to as Algorithm 2 in the sequel. (See [3], p. 6.13 .) In the next section it is seen that there is a considerable advantage to the approach of Algorithm 1, both with respect to the complexity of subsequent S-polynomials, and the extra freedom allowed during their reduction. Following this, we discuss some improvements to the scheme given in [7], which combines Algorithm 1 with multivariate factorization at cutpoint 1 (by making recursive calls to the scheme when "h" factors). An example is given of an adaptation of this scheme to a simple yet difficult class of problems (namely, dense systems of low degree).

2. Tuning the Reduction Algorithm

Clearly the cost of Algorithm 1 is dominated by the steps which perform polynomial arithmetic, namely the forming and reduction of S-polynomials. Hence it is easily seen why criteria that detect zero-reductions *a priori* are so important. Consider an ideal basis $G = [g_1, ..., g_k]$ of polynomials in an ordered set of variables, and let p be another polynomial in these indeterminates. We say that p *reduces modulo* G if the set

$$R_p \equiv \{f \in G \text{ such that } hterm(f) \mid hterm(p) \} \subset G$$

is non-empty; if p reduces (to p'), we then write

$$p \succ p' = p - [M(p)/M(g)]\, g \ , \ \text{for } g \in R_p$$

(although it is better in practice to avoid explicit fractions if working over a fraction field; see [6]). If we have $R_p = R_{p_1} = \ldots = \emptyset$, where $p_1 \equiv p - M(p)$ and $p_{i+1} \equiv p_i - M(p_i)$, then p is in normal form modulo G. Without loss of generality, let us consider only R_p. The degree of freedom offered when R_p has *many* elements poses a problem: how should we select a particular $g \in R_p$? There are two obvious possibilities: we can either choose g such that $hterm(g)$ is maximal among the headterms in R_p, or choose g such that it is minimal. If we were using the total degree ordering, the sensible choice would appear to be the latter strategy. This follows from the fact that the total degree of g gives a bound on the number of terms in g, which in turn bounds the number of term operations in the reduction. However, using the lexicographic ordering, there are several reasons why this strategy may perform poorly in practice. Consider, for example, the reduction of $p = x^2y^2z^2 + x^2y^2z + \cdots + 1$ with respect to either

$$g_1 = c_{11}x^2yz + \cdots + c_{12}y^2 + \cdots + c_{13}z^4 + \cdots + c_{14} \ , \ \text{or}$$

$$g_2 = c_{21}x + \cdots + c_{22}y^5 + \cdots + c_{23}z^{11} + \cdots + c_{24} \ .$$

If we use g_2, the result may be more complex since the correction term must be larger, and since the degree of g_2 in the subordinate variables y, z is greater. Therefore we would continue the reduction process with a more complicated object. In practice, the basis polynomials in G used for reduction are themselves the products of a reduction/elimination process; so, comparisons such as this are quite realistic. Moreover, there is no reason why g_2 should have fewer terms than g_1 (and in fact, it will often have more). Finally, since g_2 must have appeared at a later point in the algorithm, and since the coefficient growth during reduction is linear, the coefficents in g_2 may be much larger than those of g_1.

It is also clear that even though $Spoly(p, g_1)$ and $Spoly(p, g_2)$ are equivalent under the "normal selection strategy" given in [3], we should choose the former. Hence, variants of Buchberger's Algorithm which discard "redundant" polynomials (such as g_1) are at a disadvantage. When several S-polynomial pairs (equivalent under the normal selection strategy) exist, a general strategy to choose from among them should somehow take into account the degrees of the polynomials in the pair. But since the situation described above is common in practice, the S-polynomial formed from the *earliest* pair created is almost always the optimal choice. In what follows, we use this simple approach implicitly. Note, however, that this degree of freedom is lost in Algorithm 2.

Let us now examine more closely the reduction process in the case of polynomials over the rationals (i.e., all coefficient arithmetic is done over $\mathbb{Z}$). Suppose we perform a reduction

$$p \succ hcoeff(g_i)\, p \ - \ hcoeff(p)\, \frac{hterm(p)}{hterm(g_i)}\, g_i \ , \ g_i \in R_p \tag{2.1}$$

where

$$length(hcoeff(g_i)) = N', \ length(\,\|g_i\|_\infty) = N, \ g_i \text{ contains } \nu \text{ terms},$$

$$length(hcoeff(p)) = P', \ length(\,\|p\|_\infty) = P, \ p \text{ contains } \pi \text{ terms}.$$

Further suppose that the times required to multiply and add integers of lengths M, N are (c_1MN) and $c_2(M+N)$, respectively, and that the time required to multiply two terms in n variables is bounded by (c_3n). (Note that, practically speaking, the degrees of all monomials should be reasonably small.) Then the total cost of the reduction (2.1) is bounded by

$$C = \pi\,(c_1N'P) + \nu\,(c_1P'N) + \nu c_3 n + c_2\,(\nu+\pi-2)(2NP) \tag{2.2}$$

Some simple experiments reveal that for the Maple system ([5]) we have $c_1 \simeq .1\ c_2$ and $c_3 \ll c_2$. Hence, noting that $N'<N$ and $P'<P$, we have

$$C \ \sim\ 2c_2P(\pi - 2 + \nu)\,N\ . \tag{2.3}$$

If polynomial p is in fact an S-polynomial (formed from elements of G), then $\pi \sim O(\nu)$. We see that the reduction cost depends more strongly on the size of the coefficients than on the number of terms. A heuristic reduction strategy which takes this into account is the following: associate with each basis polynomial a "complexity" value, say

$$complexity(f) \ \equiv\ length(\,\|f\|_\infty)\,(\#\ of\ terms\ in\ f)^{1/2}\ , \tag{2.4}$$

and sort G in order of ascending complexity. Then for p we may choose $g \in R_p$ such that its complexity is minimal.

Let us now compare reduction strategies (in the Maple language) as follows. Consider first Algorithm 1 (which keeps all polynomials in original form), and have it reduce with respect to a basis which is sorted in descending lexicographic order, in ascending lexicographic order, or in ascending order by complexity (2.4). Then, to demonstrate the importance of using the "redundant" polynomials *at least* in forming S-polynomials, consider also Algorithm 2. This has been implemented so that *Fully_Reduce* will reduce first the smallest (lexicographically) of the polynomials which can be reduced (i.e., like the normal selection strategy for S-polynomials), and to reduce using our heuristic strategy. The following timings (and those in the sequel) were computed using Maple Version 4.0 on a VAX/785 processor. The test problems are described in the Appendix.

Problem		Algorithm 1			Algorithm 2
		ascending basis	descending basis	ascending by complexity	
1	time (sec)	570	350	298	467
	storage (Kb)	1164	1172	1106	1270
2 (Butcher)	time	646	632	657	646
	storage	1221	1222	1196	1336
3(a) (Fee (a))	time	5622	2495	1949	>136000
	storage	1786	1442	1474	**>17000**
4(a)	time	7675	12000	5698	>186000
	storage	2458	3202	2565	**>17000**
5 (Trinks)	time	10333	13753	7672	>168000
	storage	2200	2507	2392	**>17000**
6 (Katsura)	time	19490	43448	18099	**>200000**
	storage	2556	2744	2556	>14000
3(b) (Fee (b))	time	150008	182691	90688	272750
	storage	4170	7112	3580	14065
7 (Rose)	time	119770	207602	96736	72011
	storage	7600	9636	7422	8700
8	time	>72000	>90000	>83000	>92000
	storage	**>17000**	**>17000**	**>17000**	**>17000**

One reason for the superiority of the heuristic approach has not yet been mentioned: if we use for reduction a g_i with small head coefficient, the bound on the coefficient growth for the step (2.1) is also small. Hence, the [significant] cost of removing the integer content *after* the reduction should also be minimized. Typically, the reduction strategies based on headterms perform well when their correlation with the heuristic is high.

3. Using Multivariate Factorization

Despite the improvement possible with a heuristic reduction strategy, Algorithm 1 still requires long running times for (apparently) modest problems. This is particularly true when the initial extent of triangularization is small. For example, Problem 2 is relatively easy because, in a suitably chosen ordering of variables, the input polynomials are close to full separation of variables. By comparison, Problem 8 (with fewer variables) is difficult because it is more homogeneous in structure.

Fortunately, the intermediate results produced by the algorithm permit multivariate factorization surprizingly often. In [7], we presented an efficent recursive variant of Algorithm 1 to exploit this fact. This scheme (which we henceforth refer to as Algorithm 3) displayed a vast improvement in efficency compared to Algorithm 1, when both used a lexicographically descending basis in the reduction sub-algorithm. We now make some further improvements (which, implicitly, include use of the new reduction scheme), and give an example of how to exploit the extra flexibility offered by this approach. (The structure of Algorithm 3 will become clear when we present this modification.)

The results of the previous section show why Algorithm 1 was chosen as the framework of the new scheme. However, Algorithm 2 might also perform well *if* combined with a heuristic to determine when (and to what extent) the entire basis should be reduced. One obvious special case is the appearance of a univariate polynomial. When factoring is introduced, it is common for one or more univariate polynomials (usually in the base variable "x_n") to appear - at *any* point in the algorithm - when a factorization succeeds. This is so even for systems with infinitely many solutions, since these often give rise to sub-problems with finitely many solutions. We note the following (obvious) fact:

Lemma 1: Let F be a set of polynomials in $K[x_1, ..., x_n]$ which contains $p \in F \cap K[x_j]$. If p does not factor and $Ideal(F) \neq (1)$, then p is the polynomial in $Ideal(F) \cap K[x_j]$ of lowest degree.

So, if another univariate appears, the sub-problem is extraneous and can be abandoned without further reductions (or factorizations). It is also clear that a univariate polynomial should be used to reduce previous polynomials, since it is an optimal element of the reduction basis. The resulting "improved" version of Algorithm 3, although not much different, can be much faster (as we shall see).

So far, we have not exploited the fact that, when a factorization produces several sub-problems, we may apply entirely different methods to them. For example, the total degree method mentioned in Section 1 may become applicable. Or, a sudden drop in the degree of a particular variable may suggest the use of pseudo-remaindering and resultants in the manner of [9]. Since no single scheme will be viable for all problems, we will discuss one possible adaptation of Algorithm 3. This was motivated by systems such as Problem 4(b), which is only of total degree 2 in four variables, but dense. The Gröbner basis using the total degree ordering of terms is relatively easy to obtain, irespective of the density. But let us suppose that (for some reason) we wish to obtain the solutions by the lexicographic-elimination method. It happens (*because* of the density) that, during the elimination and factoring, very large extraneous factors - and hence sub-problems - are produced. These inevitably cause time/space limits to be exceeded before the more modest computation of actual solutions can proceed. However, if the total degree Gröbner basis is already known, then each factor can be tested for consistency by using it to extend the basis in the total degree ordering. In this way, only "consistent" polynomials are passed to the elimination scheme. Let *TG(F)* and *TNF(p, F)* denote the Gröbner basis of F and the normal form of p modulo F, respectivcly, in the total degree ordering. Then a hybrid scheme may be constructed as follows.

Algorithm 4:

sol_bases ← ∅ ; *append_bases*(F) , where
append_bases ← **procedure**(F, B, newpoly, nonzero, H)
  if #arguments = 1 **then**
    k ← *length*(F) ; B ← {[i,j] | 1≤i<j≤k} ; G ← F
    nonzero ← ∅ ; H ← *TG*(F)
  else
    k ← k+1 ; G ← F ∪ {newpoly} ; B ← B ∪ {[i,k] | 1≤i<k}
  while B ≠ ∅ **do**
    [i,j] ← *Select_Pair*(B, G) ; B ← B - {[i,j]}
    if *Criteria*([i,j], B, G) **then**
      p ← *NormalForm*(*Spoly*(G_i, G_j), G)
      if p ≠ 0 **then**
        P ← {distinct factors of p}
        if P ≠ {p} **then**
          for m **from** 1 **to** *length*(P) **do**
            if P_m ∈ nonzero **then next**
            q ← *TNF*(P_m, H)
            if q ≠ 0 **then**
              $\tilde{H}$ ← *TG*(H∪{q})
              if $\tilde{H}$ = {1} **then next**
            else
              $\tilde{H}$ ← H
            append_bases(G, B, P_m, nonzero ∪ {P_1,...,P_{m-1}}, $\tilde{H}$)
          return
        else
          k ← k+1 ; G ← G ∪ {p} ; B ← B ∪ {[i,k] | 1≤i<k }
  sol_bases ← sol_bases ∪ { *Fully_Reduce*(G) }
  return
end

(Note that Algorithm 3 is obtained by removing the steps involving the total degree ordering.) Here, each factor not marked as nonzero is reduced modulo H, the total degree Gröbner basis corresponding to the current (irreducible, lexicographic) sub-problem. If the factor is part of a *larger* ideal, we use its reduced form to extend H in the total degree ordering. (This will produce {1} if the factor is extraneous.) If the factor is consistent, we continue in the lexicographic ordering. Again, although it is not stated explicitly above, we should use any univariate polynomials which appear to reduce the rest of the basis. In fact, since we no longer have to consider extraneous sub-problems in the lexicographic part of the scheme, we can go much further. Define variable x_k to be *separated* if:

(a) $\exists$ irreducible $p \in F \cap K[x_k]$, or

(b) $\exists\ p \in F$ such that $hterm(p) = x_k$, and all variables in $p - M(p)$ are separated.

Then, by Lemma 1, any polynomial containing only separated variables must reduce to zero. The regular algorithm ensures separation of variables by actually carrying out an exhaustive set of S-polynomial reductions (only some of which will be avoided by the standard "criteria"), often *after* full separation has occurred. Since we are working with irreducible solution components, even the simple criterion above often allows us to avoid very large zero-reductions, when the sub-problem contains only finitely many solutions.

Let us now compare the original and improved versions of Algorithm 3 with an implementation of Algorithm 4 (including the above feature); in the latter case, the times include the separate computation of the total degree Gröbner basis.

Problem		Algorithm 3		Algorithm 4
		original	improved	(hybrid)
1	time (sec)	297	290	266
	storage (Kb)	1286	1488	1286
2	time	362	263	3774
(Butcher)	storage	1393	1335	1630
3(a)	time	885	665	806
(Fee (a))	storage	1786	1712	1622
4(a)	time	3460	1962	2650
	storage	3031	2572	2448
5	time	2354	1753	1902
(Trinks)	storage	2220	2032	1933
6	time	4386	3028	2810
(Katsura)	storage	2712	2433	2253
3(b)	time	22773	11241	**>200000**
(Fee (b))	storage	3048	2842	>4400
7	time	14340	8167	12464
(Rose)	storage	3876	4260	3612
8	time	191132	78045	64930
	storage	11592	11076	7208
4(b)	time	>168000	>141000	65314
	storage	**>17000**	**>17000**	10028

The expected asymptotic improvement of Algorithm 3 over Algorithm 1 is clear. We also note that Algorithm 4 is the only scheme which is able to cope with the dense problem 4(b). This is of interest because most problems tend to become denser as the first few variables are eliminated. It is surprizing that Algorithm 4 is competitive on many of the simpler problems, since use of the total degree ordering to "filter" extranneous factors is less than an ideal general solution. Although modular methods for Gröbner bases are elusive (see [8]), the development of such an algorithm for this specific task seems a worthwhile problem for future study. It should also be possible to weaken somewhat our definition of "full separation of variables".

Appendix: List of test problems

Problem 1 :

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

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

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

using $x > y > z$.

Problem 2 : Butcher's system (see [1]) of 8 equations in 8 unknowns, using the ordering $b_1 > a_{32} > b_2 > b_3 > a > c_3 > c_2 > b$.

Problem 3 : Fee's system (see [6]) of 4 equations in 5 unknowns.

(a) Substitute b=2, and use the poor ordering $d>q>p>c$.

(b) Use the ordering $c>d>q>b>p$.

Problem 4:

$$F_1 = 4 + 8w - 10w^2 - 10z + 7zw - 3z^2 - 3y + 6yw + 2yz + c_1y^2 - 9x + 5xw - 2xz - 4xy + d_1x^2,$$

$$F_2 = 9 - 6w + 6w^2 - 2z + 10zw - 5z^2 + 7y + yw - 2yz + c_2y^2 - 9x - 8xw - 4xz - 8xy + d_2x^2,$$

$$F_3 = -9 - 2w + 10w^2 - 9z + 8zw + 10z^2 - 3y - 2yw - 2yz + c_3y^2 - 2x + 3xw - 9xz + 7xy + d_3x^2,$$

$$F_4 = -7 + 9w + 2w^2 + 3z + 10zw - 2z^2 + 8y + 4yw - 3yz + c_4y^2 + 3x - 6xw + 9xy + d_4x^2 .$$

(a) Substitute $c_1 = c_2 = c_3 = c_4 = d_1 = d_2 = d_3 = d_4 = 0$, and use the ordering $x>y>z>w$.

(b) Substitute c_1 = -8, c_2 = 6, c_3 = -3, c_4 = -6, d_1 = -1, d_2 = -5, d_3 = 6, d_4 = -3, and use $z>x>w>y$.

Problem 5 : Trinks system [10], using the poor variable ordering $b>t>s>w>p>z$.

Problem 6 : Katsura's system (see [1]) of 5 equations in 5 unknowns, using the ordering $u_4 > u_0 > u_3 > u_2 > u_1$.

Problem 7 : Rose's system (see [1]) of 3 equations in 3 unknowns, using the ordering $u_4 > u_3 > a_{46}$.

Problem 8:

$$F_1 = 4 + 8z - 10z^2 - 10z^3 + 7y - 3yz - 3yz^2 + 6y^2 + 2y^2z - 8y^3 - 9x$$
$$+ 5xz - 2xz^2 - 4xy - xyz + 9xy^2 - 6x^2 + 6x^2z - 2x^2y \ ,$$
$$F_2 = -5 + 7z + z^2 - 2z^3 + 6y - 9yz - 8yz^2 - 4y^2 - 8y^2z - 5y^3 - 9x$$
$$- 2xz + 10xz^2 - 9xy + 8xyz + 10xy^2 - 3x^2 - 2x^2z - 2x^2y \ ,$$
$$F_3 = -2 + 3z - 9z^2 + 7z^3 + 6y - 7yz + 9yz^2 + 2y^2 + 3y^2z + 10y^3$$
$$- 2x + 8xz + 4xz^2 - 3xy - 6xyz + 3xy^2 - 6x^2 + 9x^2y \ ,$$

using x>y>z .

References

[1] W. Böge, R. Gebauer, H. Kredel: "Some Examples for Solving Systems of Algebraic Equations by Calculating Gröbner Bases", J. Symbolic Computation, Vol. 2, No. 1, 1985.

[2] 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.

[3] 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.

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

[5] B.W. Char, G.J. Fee, K.O. Geddes, G.H. Gonnet, M.B. Monagan: "A Tutorial Introduction To Maple", J. Symbolic Computation, Vol 2, No. 2, 1986.

[6] S.R. Czapor, K.O. Geddes: "On Implementing Buchberger's Algorithm For Gröbner Bases", Proc. SYMSAC '86, (B.W. Char, ed.), Waterloo, July 1986, pp. 233-238.

[7] S.R. Czapor: "Solving algebraic equations: combining Buchberger's algorithm with multivariate factorization", submitted to J. Symbolic Computation, Sept., 1986.

[8] G.L. Ebert: "Some Comments On The Modular Approach To Gröbner Bases", ACM SIGSAM Bull. 17, No. 2, May, 1983.

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

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