J. Symbolic Computation (1997) **11**, 1–000

Computing the Dimension of a Projective Variety: the Projective Noether Maple Package

MARC GIUSTI[†], KLEMENS HÄGELE[‡], GRÉGOIRE LECERF[†],
JOËL MARCHAND[†], BRUNO SALVY[*]

[†]*Laboratoire GAGE, École polytechnique, F-91128 Palaiseau, France*
[‡]*Matemáticas, Universidad de Cantabria, E-39071 Santander, Spain*
[*]*Projet ALGO, INRIA Rocquencourt, F-78153 Le Chesnay Cedex, France*

(Received 30 December 1997)

Recent theoretical advances in elimination theory use non-classical data structures to represent multivariate polynomials. We present the *Projective Noether Package* which is a Maple implementation of a series of these new algorithms for the computation of the dimension of a projective variety. The package contains hybrid code mixing the old and new approaches, thus allowing to investigate the potential of the new techniques. We present a comparison of the more traditional algorithms already available within Maple with the new ones, trying several possible strategies. Comparative results on benchmarks for time and space of three different families of multivariate polynomial equation systems are given and we point out both weaknesses and advantages of the different approaches.

Keywords: Benchmark, Maple, polynomial equation system, Noether position, straight-line programs.

1. Introduction

Classical methods to study and solve systems of polynomial equations are based on numerous avatars of Gröbner (or standard) bases algorithms or Riquier-Janet type methods (Ritt-Wu's algorithm). All these methods use implicitly but deeply the dense or sparse representation of multivariate polynomials, which is the computer science counterpart of the expansion of these mathematical objects on the monomial basis of the polynomial algebra. And all these methods can be interpreted as *rewriting* techniques.

Considerable efforts have been made in order to improve both theoretical and practical aspects of these techniques and to produce efficient algorithms and implementations. Restricting to Gröbner bases algorithms and to quote only the most commonly available packages, see the Maple `grobner package` (Char *et al.* 1991), `axiom` (Watt *et al.* 1995), `gb` (Faugère 1995), `singular` (Greuel *et al.* 1997), `macaulay2` (Stillman and Bayer 1996), `reduce` Hearn (1987). Some very recent benchmarks can be found in (Faugère 1997).

The knowledge of a standard basis yields as a simple byproduct the dimension of the algebraic variety defined by such systems. Actually, one can show that focusing on the simpler problem of computing the dimension of a projective algebraic variety will lead to

0747–7171/90/000000 + 00 $03.00/0

a better worst-case complexity than the whole construction of a standard basis (Giusti 1988).

We consider the *unit cost measure* model, i.e., each arithmetic operation of the ground field is counted as one. Then the complexity is polynomial in the size of the intermediate expressions computed. Its seems that there is no hope to design an algorithm whose complexity is *polynomial* in the size of the input, since the intermediate computations are not. But this observation is only valid if we stay stuck in the dense representation context. A breakthrough was obtained by Giusti and Heintz (1993), resulting in the existence of an algorithm with polynomial behaviour, provided one uses a mixed representation for intermediate computations.

Actually the algorithm described in *loc. cit.* computes a little bit more, i.e., a change of coordinates putting the new variables in *Noether position*. Informally speaking, the variables are separated into two subsets of different nature: the independent and dependent ones. Furthermore the polynomials occurring as intermediate computations are coded in the dense representation w.r.t. the dependent variables, while their coefficients are polynomials in the independent ones coded by *arithmetic circuits*, also called *straight-line programs*. This means that these latter polynomials are represented by programs evaluating them at a point (of the ground field) using only additions and multiplications (of the ground field). We will conveniently refer respectively to the *writing data structure* and the *evaluation data structure*.

Mixing these two data structures was successfully used in a series of papers to design a new geometric elimination algorithm (see in the references the joint works by Giusti, Hägele, Heintz, Montaña, Morais, Morgenstern, Pardo 1995–97). An efficient implementation of the complete elimination algorithm will require some time to collect more practical experience with the first experimental prototypes for some of its components. Some basic but important steps were made already, but there is still a lot of work left (on the use of evaluation data structures, see the works by Aldaz Zaragüeta, Castaño, Hägele, Llovet, Martínez, Matera within the Tera project `http://hilbert.matesco.unican.es/tera`).

We present here another step, a Maple program called the *Projective Noether Package* implementing the algorithm from (Giusti and Heintz 1993). (The package and its documentation are available at `http://medicis.polytechnique.fr/tera/soft.html`). It turns out that a not so well-known functionality of Maple is the systematic use of the evaluation data structure, which is described in the first part of this article. Consequently we can compare the more traditional algorithms already available within Maple with the new ones, experimenting with several possible strategies. The package contains hybrid code mixing the old and new approaches, thus allowing to investigate the potential of the new techniques. Comparative results on benchmarks for time and space of three different families of multivariate polynomial equation systems are given and we point out both advantages and weaknesses of the different approaches. One of the most encouraging results is provided by an example where our Maple implementation computes an upper bound for the dimension more than fifty times faster than the `gb` system.

Acknowledgments. This work was supported in part by the Long Term Research Project Alcom-IT (#20244) of the European Union, the French and Spanish grants: GDR CNRS 1026 MEDICIS and PB93–0472–C02–02.

2. Evaluation data structure and Maple implementation

2.1. Directed Acyclic Graphs and Straight-line Programs

Let k be an infinite effective field; this means that the arithmetic operations (addition, subtraction, multiplication, division) and basic equality checking (comparison) between elements of k are realizable by algorithms. All algorithms below will be represented by *arithmetic networks* over k i.e., directed acyclic graphs (DAG's) whose internal nodes are labelled by arithmetic operations of k, by Boolean operations corresponding to propositional logic, and by selectors associated with equality checking of elements of k. The external nodes of the graph represent the inputs and the outputs of the network. The inputs are always elements of k and the outputs may be elements of k, Boolean values, or integers of limited range (represented by vectors of Boolean values).

Particular arithmetic networks are of special interest: *arithmetic circuits* or *straight-line programs* (SLP's), without divisions nor branching, containing neither selectors nor (propositional) Boolean operations. Generally speaking the *size* of the DAG (or the *sequential complexity* of the arithmetic network) is nothing but the number of its nodes (thus for a SLP the number of additions and multiplications involved). For details and elementary properties of the notion of straight-line programs we refer to Strassen (1972), von zur Gathen (1986), Stoß (1989) or Heintz (1989).

Let $x_1, \ldots, x_n$ be indeterminates over k. A polynomial of $k[x_1, \ldots, x_n]$ is usually coded in the so-called *dense representation* as the vector of its coefficients. But straight-line programs can also be used to code a multivariate polynomial, computing its value at a point of k^n.

The only non-trivial point when dealing with this data structure is equality checking (or zero testing). We shall perform this task by choosing a suitable "correct test sequence" of points with coordinates from k according to a theorem of Heintz and Schnorr (1982), which we recall for completeness:

Let D and L be two positive integers, and let us define the subset $W(D, n, L)$ of polynomials of $k[x_1, \ldots, x_n]$, of degree at most D, which can be coded by a SLP with at most L arithmetic operations. Let us consider $m := 6(L+n)(L+n+1)$, and a family $\gamma := \{\gamma_1, \ldots, \gamma_m\}$ of m points in k^n. Such a set is a *correct test sequence* for $W(D, n, L)$ if every polynomial in the latter vanishing on the points of γ is actually identically zero.

Theorem 2.1. (Heintz and Schnorr (1982), Theorem 4.4) *With the notations above let us fix a subset Γ of k of cardinality $\#\Gamma = 2L(D+1)^2$. The subset $\tau(D, n, L, \Gamma) \in \Gamma^{nm}$ of correct test sequences for $W(D, n, L)$ satisfies:*

$$\#\tau(D, n, L, \Gamma) \geq (\#\Gamma)^{nm}(1 - (\#\Gamma)^{-\frac{m}{6}}).$$

Although the choice of such a correct test sequence could be done algorithmically, the cost of doing so would exceed the main complexity class we want. Therefore the algorithms we study below will be non-uniform insofar as they depend on the choice of correct test sequences. On the other hand, the theorem allows us to randomly choose correct test sequences with a probability of failure which becomes arbitrarily small as the parameters D, n and L increase. Therefore our algorithms can be uniformly randomized within the same order of (average) complexity. In doing so we encounter the following kind of probabilistic procedure which we call a *randomized algorithm.* A randomized algorithm has error probability bounded by some $0 \leq \varepsilon < 1/2$ when accepting an input and error probability zero when rejecting it (in our case we may choose $\varepsilon = 1/262144$). As far as our algorithms compute polynomials or rational functions the correctness of the output depends only on the correctness of previous and intermediate decisions made by probabilistic procedures and can be checked randomly. In this sense, we apply also the term *randomized procedure* to the computation of polynomials or rational functions. Thus our results are valid not only in the sense of the non-uniform complexity model, but also in the sense of probabilistic (randomized) algorithms (see Balcázar *et al.* (1995), § 6.6, Giusti and Heintz (1993), § 1.2.3, § 1.3 and § 2.2 and Fitchas *et al.* (1995), § 1.3 and § 2.1 for more details).

To sum up we get the weakened following form which allows a probabilistic treatment of the theorem:

THEOREM 2.2. (FITCHAS *et al.* (1995), THEOREM 2.1) *There exists an arithmetic network over k of size $O(Lm) = O(L(L+n)^2)$, which given any SLP (without divisions) of size at most L, checks if the n-variate polynomial it represents is identically zero. Moreover the network can be constructed by a probabilistic algorithm in sequential time $O(L(L+n)^2)$ with a probability of failure uniformly bounded by $\varepsilon := 1/262144$.*

2.2. Efficient Evaluation in Maple

The Maple computer algebra system is based on a systematic use of common subexpression sharing. Objects which might look like expression trees to the user are actually stored as directed acyclic graphs, where only one copy of each distinct subtree is kept. This is realized by maintaining a hash table of all the expressions occurring simultaneously in a session. The structure thus obtained can be viewed as a single directed acyclic graph the children of whose root correspond to all the distinct subexpressions residing simultaneously in memory. A simple consequence of this representation is that syntactic equality of expressions is reduced to checking equality of addresses and can thus be performed in constant time. This provides the basis for the efficiency of Maple's *option remember* which can be used to record the pairs (input, output) of a procedure. When the procedure contains a recursive call to itself on the subexpressions of its argument, it then performs a DAG traversal of the expression, instead of a tree traversal without this option. This can lead to an improved complexity of the algorithm.

For instance, it is unfortunate that up to the current version (V.4), Maple's `subs` command, which is commonly used to evaluate an expression at a point, does not benefit from this nice mechanism and has a complexity related to the size of the tree instead of the size of the DAG. The use of a DAG traversal to improve the complexity can be illustrated with a simple alternate procedure:

```
dagsubs := proc(tosubs::{name = algebraic, list(name = algebraic)}, expr)
    local dosubs, i;
        dosubs := proc(expr) option remember;
                if 1 < nops(expr) then map(procname, expr) else expr fi end;
        if type(tosubs, name = algebraic) then dosubs(op(1, tosubs)) := op(2, tosubs)
        else for i in tosubs do dosubs(op(1, i)) := op(2, i) od fi;
        dosubs(expr)
    end
```

Table 1. Substitution and DAG's

n	30	31	32	33	34
`subs`	10sec	16sec	26sec	38sec	62sec
	46Mb	76Mb	120Mb	199Mb	321Mb
`dagsubs`	5sec	7sec	17sec	24sec	43sec
	25kb	26kb	28kb	28kb	29kb

In Table 1, we give examples of the time and memory† required by both `subs` and this simple `dagsubs`. The test suite is the following sequence of polynomials:

$$P_0(x) = 1, \qquad P_1(x) = x, \qquad P_{n+2}(x) = xP_{n+1}(x) + P_n(x) + 1, \quad n > 0,$$

where x is replaced by another variable y. The time difference is not very large, but `subs` is a function of Maple's compiled kernel, whereas `dagsubs` is interpreted. However, while `dagsubs` needs a very limited amount of memory, `subs` requires more than one thousand times this amount, and the ratio increases very fast with the index of the polynomials. This example clearly demonstrates that working with DAG's when possible can be crucial in terms of efficiency.

When the DAG is large and many evaluations of it at different values are required, it is possible to convert this DAG into a so-called *computation sequence*, which is a Maple variety of straight-line programs, with nodes of arbitrary arity. This computation sequence is obtained by Maple's `optimize` command. For instance, on the polynomial P_{10} above, one obtains:

$$t_1 = x^2, \quad t_3 = x(t_1 + 2), \quad t_5 = x(t_3 + x + 1), \quad t_7 = x(t_5 + t_1 + 3),$$

$$t_9 = x(t_7 + t_3 + x + 2), \quad t_{11} = x(t_9 + t_5 + t_1 + 4), \quad t_{13} = x(t_{11} + t_7 + t_3 + x + 3),$$

$$t_{18} = x(x(t_{13} + t_9 + t_5 + t_1 + 5) + t_{11} + t_7 + t_3 + x + 4) + t_{13} + t_9 + t_5 + t_1 + 6.$$

This can then be translated into a Maple procedure or alternatively into Fortran or C code by the `‘optimize/makeproc‘`, `Fortran` or `C` commands. Here is for instance the corresponding Maple procedure:

```
proc(x)
local t1, t18, t13, t11, t9, t7, t5, t3;
    t1 := x^2;
    t3 := x*(t1 + 2);
    t5 := x*(t3 + x + 1);
    t7 := x*(t5 + t1 + 3);
    t9 := x*(t7 + t3 + x + 2);
    t11 := x*(t9 + t5 + t1 + 4);
    t13 := x*(t11 + t7 + t3 + x + 3);
    t18 := x*(x*(t13 + t9 + t5 + t1 + 5) + t11 + t7 + t3 + x + 4) + t13
```

† The tests in this article have been performed on `pierre.polytechnique.fr`, a PC Pentium Pro (200Mhz) with 512Mb of memory, running Linux 2.0.27 and Maple V.3. This computer forms part of the equipment of GDR Medicis: `http://medicis.polytechnique.fr`.

```
          + t9 + t5 + t1 + 6
end
```

The Fortran and C routines are entirely analogous.

2.3. Example: testing the regularity of a matrix

In this section, we illustrate how the systematic exploitation of the DAG structure leads to improved algorithms checking whether square matrices of multivariate polynomials with integer coefficients are singular or not. This task turns out to be a basic building block of the algorithm developed in the next section.

The matrices we take in our examples are square $k \times k$ matrices with polynomial entries having between 3 and 9 variables. The matrices are sparse with $5k$ entries at random filled by polynomials provided by Maple's `randpoly` function. To obtain regular matrices, the diagonal is first filled with 1's. To obtain singular matrices, the columns from 1 to $k-1$ are summed into the kth one. Timings for regular and singular matrices turn out to be similar, thus we give only one table (Table 3) of results.

Naive approach

The simplest idea is to compute the determinant of such a matrix using for instance Maple's `det` command, which is based on a mixture of fraction free Gaussian elimination and minor expansion. The matrix is singular if and only if the resulting polynomial is 0. Since the resulting polynomial is always expanded, recognizing zero is very easy, but the computation itself is expensive because of the exponential growth of the number of multivariate monomials as the degree increases. This shows in Table 3, since on all of our examples, Maple returns an error message "object too large" (abbreviated 'otl' in the table).

Straight-line programs and Berkowitz's algorithm

In order to overcome the exponential complexity of expanding determinants, it is natural to turn to DAG's or to straight-line programs evaluating them. Recognizing zero with this data-structure becomes the expensive operation. Thus an approach based on Gaussian elimination does not apply anymore. Several other algorithms can be applied. We use an algorithm due to Berkowitz (1984) which has the advantage of a simple description. Given an $n \times n$ matrix, it computes its characteristic polynomial (and in particular its determinant) in $O(n^4)$ arithmetic operations on the coefficients and requires neither test nor division.

On a generic square matrix of size n, the number of nodes of the DAG evaluating the

Table 2. Sizes of different representations of the determinant of an $n \times n$ matrix

dimension	2	3	4	5	6	7	8	9
dag size (expanded)	3	7	25	121	721	5041	40321	362881
dag size (Berkowitz)	3	20	64	169	343	664	1104	1817

Table 3. Computations on matrices with polynomial entries

dimension	10	20	50	100	200
naive	otl	otl	otl	otl	otl
Berkowitz	.08	1.5	400	> 5000	
test sequence (#pts)	$5.7\,10^7$	$2.3\,10^{10}$	$8.0\,10^{13}$		
DAG `subs`	53	> 5000			
DAG `optimize`	.02	> 5000			
direct Maple	.03	.20	23	47	1050

expanded determinant and the one produced by Berkowitz's algorithm are indicated in Table 2. In this case, the polynomial complexity of Berkowitz's algorithm quickly yields better results than the number $n! + 1$ of monomials of the generic determinant. This is naturally reflected by the time required for the computation. For our test matrices, the results turn out to be very similar: Berkowitz's algorithm takes almost no time on matrices for which `det` cannot compute the result. This appears in the second line of Table 3.

Evaluation and test sequences

In order to prove that the matrix is regular from the DAG computed by Berkowitz's algorithm, it is sufficient to find a point where the evaluation of this DAG yields a non-zero result. In practice, any random point will do. When the matrix is actually singular, we rely on correct test sequences as explained in §2.1.

In line 4 of Table 3, we give the time used to evaluate the DAG computed via Berkowitz's algorithm at one point. In regular cases, this is usually also the time required to prove that the matrix is regular. In the singular cases, we also show an estimate of the number of points m forming the correct test sequences for DAG's of the corresponding size. The table shows that although this approach makes it possible to deal with objects which are too large for Maple when expanded, it also rapidly produces objects with which it is impossible to proceed in a reasonable amount of time. This might be due to the limited size of the hash tables used by Maple. Whatever the speed of evaluation, the number of points indicated on line 3 leads to the conclusion that the theoretical bound which leads to polynomial complexity is much too large to be of practical use.

Direct evaluation

The process outlined above consists of two steps. First a DAG is constructed via Berkowitz's algorithm, then this DAG is evaluated at one or several points. It is then a natural strategy to first evaluate the matrix at these points and then compute the determinant. In the regular case this is clearly an improvement, since the DAG for the determinant does not need to be stored in memory anymore and the determinant can be evaluated in $O(n^3)$ arithmetic operations. In the singular case, this approach also works because we can still use the bound on the number of points which follows from considering the DAG produced via Berkowitz's algorithm without actually executing this algorithm. Thus we use the theoretical results on the complexity of a straight-line program based approach to guide the practical computation, without actually computing with straight-line programs. The resulting timings appear in the last line of Table 3 and vindicate this strategy. Preliminary experiments in C using LEDA's bignums (Mehlhorn *et al.* 1997) seem to indicate that we can only expect an improvement of a factor four to five by performing this evaluation directly in C.

3. Computing the dimension of an algebraic variety

3.1. Noether position

Let k be an infinite effective field, and $\overline{k}$ be an algebraic closure of k. Given a set of homogeneous polynomials $f_1, \ldots, f_s$ in $k[x_0, \ldots, x_n]$, consider the projective variety $V = V(f_1, \ldots, f_s)$ generated by the f_i in projective n-space $\mathbb{P}^n_{\overline{k}}$. We want to calculate the dimension of the projective variety V.

There are several approaches to this problem. We distinguish two different tasks, first to give some upper bound; and second to certify that an integer known to be an upper bound is actually the dimension by a deterministic or probabilistic algorithm.

Giusti and Heintz (1993) gave an algorithm to compute the dimension, which actually computes a change of coordinates putting the new variables in Noether position.

The variables $x_0, \ldots, x_r$ are said to be *independent* with respect to V if $(f_1, \ldots, f_s) \cap k[x_0, \ldots, x_r]$ is the trivial ideal (0). If moreover the canonical homomorphism

$$k[x_0, \ldots, x_r] \to k[x_0, \ldots, x_n]/(f_1, \ldots, f_s)$$

is an integral ring extension, the variables $x_0, \ldots, x_n$ are said in *Noether position* with respect to V. The latter condition means that the canonical images of the variables $x_{r+1}, \ldots, x_n$ satisfy integral dependence relations (in other words are algebraic integers) over $k[x_0, \ldots, x_r]$. As a consequence the dimension of V is nothing but r. In order to simplify the complexity considerations we shall suppose that d is at least n.

Theorem 3.1. (Giusti and Heintz (1993)) *Let $f_1, \ldots, f_s$ be homogeneous polynomials of degree at most d in $k[x_0, \ldots, x_n]$, defining a projective variety V in $\mathbb{P}^n_{\overline{k}}$. There exists a randomized algorithm without divisions which computes with sequential complexity $s^{O(1)}\, d^{O(n)}$ a linear change of coordinates over k such that the new variables are in Noether position with respect to V.*

Let us recall the main steps of this algorithm. It is organized around a loop, with decreasing index m starting from n. The $(n-m)$th iteration is entered with the condition: the images of $x_{m+1}, \ldots, x_n$ are already algebraic integers over $R^{(m)} := k[x_0, \ldots, x_m]$.

- Let z be a new variable; we denote by $f_i^{(m)}$ the polynomial
$$f_i(zx_0, zx_1, \ldots, zx_m, x_{m+1}, \ldots, x_n).$$
This is an homogeneous polynomial in $k[x_0, \ldots, x_m][x_{m+1}, \ldots, x_n, z]$.
- Let W be the projective variety in $\mathbb{P}^{n-m}_{\overline{K}}$ defined by $f_1^{(m)}, \ldots, f_s^{(m)}$, where $K = K^{(m)}$ is the field of fractions $k(x_0, \ldots, x_m)$ of $R^{(m)}$.
- In this situation a *criterion for independence* is that W is not empty, which can be checked by computing a Gröbner basis or by applying an effective projective Nullstellensatz.

- If the criterion is satisfied, we are in Noether position and we stop.
- Otherwise, there exists a non-zero homogeneous polynomial g in $k[x_0, \ldots, x_m]$, which both techniques can exhibit, and a point of $\mathbb{P}^n{}_k$ with homogeneous coordinates $(a_0, \ldots, a_m)$ (with $a_m \neq 0$) on which g does not vanish. After the change of coordinates $x_0 \leftarrow x_0 + a_0 x_m, \ldots, x_{m-1} \leftarrow x_{m-1} + a_{m-1} x_m, x_m \leftarrow a_m x_m$ the polynomial g becomes monic in x_m, hence we can enter the $(n+1-m)$th iteration.

3.2. The Projective Noether Package

The above mentioned algorithm still leaves some choices for different implementation strategies, especially the order of evaluation and specialisation and the option of mixing the straight-line program ideas with the more traditional Gröbner bases techniques. We now describe the different strategies implemented in the *Projective Noether Package*:

1 The pure SLP strategy, denoted [BDE], implemented directly as proposed in (Giusti and Heintz 1993), consists of three main steps:

[B] Let $N = 1 + \sum_{i=1}^{s} (\deg(f_i) - 1)$ (the bound of the effective projective Nullstellensatz). Create Q, the matrix of the linear application $(h_1, \ldots, h_s) \mapsto h_1 f_1^{(m)} + \cdots + h_s f_s^{(m)}$, h_i being homogeneous polynomials of degree $N - \deg(f_i)$ in $R^{(m)}[x_{m+1}, \ldots, x_n, z]$. The coefficients of Q are in $R^{(m)}$.

[D] Use Berkowitz-Mulmuley linear algebra (Berkowitz (1984), Mulmuley (1987)) to construct the DAG corresponding to the determinant of the product matrix $Q^t Q$ to check the surjectivity of Q (this determinant lives in $R^{(m)}$).

[E] Using the Maple functions `optimize` and `makeproc`, the DAG is transformed into a Maple procedure. A probabilistic test is then performed to determine whether this DAG represents zero or not.

2 The evaluation and determinant strategy, denoted [ED], is organised as a loop on a set of values assigned successively to $x_0, ..., x_m$. At each step we

[E] specialize the $f_i^{(m)}$ with the values given to $x_0, ..., x_m$;

[D] build the matrix Q as before, but then use the Maple determinant function to decide whether the linear application associated with Q is surjective or not. Note that Q has now integer coefficients.

3 The evaluation and Gröbner basis strategy [EG] replaces the determinant step [D] of the previous strategy by a Gröbner basis computation:

[E] Specialize the $f_i^{(m)}$ for a given set of values for $x_0, ..., x_m$;

[G] Apply a Gröbner basis algorithm with total degree ordering to the specialized $f_i^{(m)}$ to compute a standard basis and then determine whether the variety W is empty or not.

4 The last strategy [EG-gb] is an optimized version of [EG], with the Gröbner basis computation being handled by the `gb` program of Faugère (1995) called via `gblink` (Lecerf and Schost 1997), instead of Maple's `grobner` package.

3.3. Families of Systems of Polynomial Equations

We present here three families of polynomial equation systems which we have used to compare the different strategies discussed above.

Table 4. Lower bounds for the dimension of the system `infcyclic` H_n^∞

n	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
$r_{\mathrm{inf}}(n)$	-1	-1	-1	0	0	1	1	2	3	3	4	5	5	6	7	8	8

The `sac` *polynomial system*

This system is inspired by an example taken from (Pardo 1995). The affine system can be interpreted as a way of writing how many numbers can be represented with n bits, the first k bits being fixed. The actual system used here is a homogenized, slightly twisted version of the initial one. It depends on two parameters n, k and has $k+1$ homogeneous equations of degree 2 in $n+1$ variables. The dimension of the system is $(n-k-1)$:

$$f_j = x_j^2 - x_j x_0 \quad \text{for} \quad 1 \le j \le k, \qquad f_{k+1} = \sum_{j=0}^{n} 2^j x_n x_j.$$

The `homcyclic` *and* `infcyclic` *polynomial systems*

These systems are related to the question of finiteness and structure of the corresponding set of cylic n-roots (Björck 1990). Let $x_1, \ldots, x_n$ be variables and M_i be the monomial $x_1 \cdots x_i$. Let σ be the cycle $(1, 2, \ldots, n)$ of the nth permutation group. We define the ith cyclic equation of the nth system as:

$$H_n^i = \sum_{k=0}^{n-1} \sigma^k(M_i) \quad \text{for} \quad 1 \le i \le n-1, \qquad H_n^n = M_n.$$

Now, `homcyclic` n and `infcyclic` n define the systems:

$$\texttt{homcyclic } H_n = \begin{cases} H_n^n &= x_0^n, \\ H_n^{n-1} &= 0, \\ &\vdots \\ H_n^1 &= 0. \end{cases} \qquad \texttt{infcyclic } H_n^\infty = \begin{cases} H_n^n &= 0, \\ H_n^{n-1} &= 0, \\ &\vdots \\ H_n^1 &= 0. \end{cases}$$

For example, when $n = 3$ the system H_3^∞ is $\{x_1x_2x_3 = 0, x_1x_2 + x_2x_3 + x_3x_1 = 0, x_1 + x_2 + x_3 = 0\}$.

Proposition 3.1. (in collaboration with E. Schost) *The system* `infcyclic` H_n^∞ *defines in* $\mathbb{P}^{n-1}(\mathbb{C})$ *a projective variety of dimension at least:*

$$r_{\mathrm{inf}}(n) = n - \left\lceil \frac{n}{\lfloor \sqrt{n} \rfloor} \right\rceil - \lfloor \sqrt{n} \rfloor.$$

Proof. Let $k = \lfloor \sqrt{n} \rfloor$, $l = \left\lceil \frac{n}{\lfloor \sqrt{n} \rfloor} \right\rceil$. Assume $x_1, x_{1+k}, \ldots, x_{1+(l-1)k}$ to be zero. Thus there remain $n - l$ non zero variables. And only the $k-1$ first equations are non zero, thus applying Krull's Lemma (Eisenbud 1995, 8.2.2) completes the proof. $\square$

The resulting lower bounds for the dimension of the system `infcyclic` H_n^∞ are given in Table 4.

3.4. Experimental data

We present the tables of experimental results obtained for the two tasks of bounding and computing the dimension of the families of polynomial equation systems from §3.3. The different strategies (`BDE`, `ED`, `EG` and `EG-gb`) introduced in §3.2 are applied and also compared with Maple's `grobner` function (`g`) and with `gb` applied directly to the systems. Time is measured in *seconds* and memory space in *kilobytes*. A time t preceded by the symbol $>$ means that the computation has been manually aborted after t seconds, the corresponding value in the column labelled `space` is the memory allocated until then. Empty entries in the tables are due to technical problems measuring very small quantities and/or their scaling with the rest of the entries in the same table.

Bounding the Dimension

The first task is the determination of upper bounds on the dimension of the variety. Using the algorithm of §3.1, we know that at step m of the iteration the dimension of the variety is at most $m-1$. It is also possible to compute upper bounds on the variety during the calculation of the Maple `grobner` function: the dimension is at most the dimension of the monomial ideal generated by the leading monomials of the syzygies computed when performing a Buchberger algorithm, see e.g., (Cox *et al.* 1997).

We compared the following strategies: `EG` refers to the strategy presented in §3.2, `g` refers to the Maple `grobner` function, with total degree ordering on the variables, and `gb` refers to the `gb` program (Faugère 1995) applied directly. The `gb` computation uses the function `sugar`, with the optional parameter `"info"`, and a total degree ordering. The abbreviations used here are: UBD — an upper bound on the dimension, `infcyclic n` — the polynomial system introduced in §3.3. For example the first line of Table 5 reads: the dimension of the system `infcyclic 6` has been proved to be at most 2 in 0.2 seconds with a memory space of 720kb, using the strategy `EG`, whereas using the Maple `grobner` function it took 0.71 seconds and 1180kb.

Comment on Table 5

The computation of upper bounds on the dimension of the projective variety is the strong point of the `EG` method. Where the Gröbner bases algorithms are forced to work with the complete equation system in many variables, our recursive approach yields the correct answer a lot faster. First, it can be seen that the `g` strategy corresponding to Maple's `grobner` function is the least efficient approach. We observe then, that even using this same Maple `grobner` function for the intermediate calculations in the `EG` method, the resulting performance is already competitive with the stand-alone `gb` program. Note that any improvements in the field of Gröbner bases computations will also yield direct improvements for the `EG` and `EG-gb` strategies.

Comment on Table 6 and Figures 1, 2

Even though it is possible to decompose by hand the polynomial equation system `infcyclic 8` and thus obtain better time/space results, the same is not true for its twin `homcyclic 8`. Table 6 and Figures 1 and 2 illustrate the very similar time/space behaviour of `gb` for both systems, thus showing that the decomposability of the `infcyclic` family does not influence excessively the results. The reason for us to use the `infcyclic` system for our tests is the known lower bounds for the dimension from Table 4.

Table 5. Benchmarks for the `infcyclic` systems

system / method	EG		g		gb	
UBD	Time	Space	Time	space	Time	space
`infcyclic 6`						
2	0.2	720	0.71	1180		
1	2.5	1400	8.48	1500	1.0	1200
`infcyclic 7`						
4			1.7	1400		
3	0.2	720	2.5	1500		
2	2.5	1440	65	2230	7	2200
1	900	3200	13000	6200	950	3200
`infcyclic 8`						
5			5.8	1500		
4	0.3	720	6.18	1500		
3	3.0	1500	396	2700	3	2200
2	1400	4130	$>$ 250000	$>$ 18000	54000	27600
`infcyclic 9`						
6			21	1900		
5	0.3	720	22	1900		
4	5.1	1700	2293	3080	3	3400
3	1600	7000	$>$200000	$>$ 8000	$>$ 108000	$>$ 180000

Table 6. `homcyclic 8`

`homcyclic 8`	EG-gb		gb	
UBD	Time	Space	Time	Space
5	1			
4	2.4			
3	11.2		3	2200
2	1300	4400	50000	20000

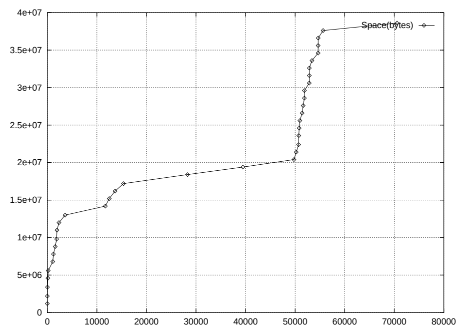

Figure 1. `homcyclic 8`

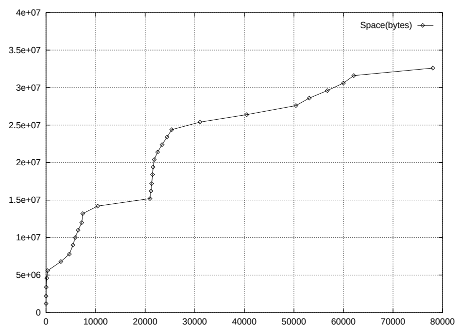

Figure 2. `infcyclic 8`

Table 7. Benchmarks for the `sac`, `homcyclic` and `infcyclic` systems

		BDE		ED		EG		g	
System	Dim.	Time	Space	Time	Space	Time	Space	Time	Space
`sac`									
2.1	0	0.13	518	0.06	297	0.11	479	0.03	168
3.1	1	0.16	675	0.30	820	0.32	1054	0.04	189
3.2	0	20539	186153	1.02	3491	0.37	1202	0.12	527
4.1	2	0.22	750	0.25	861	0.44	1229	0.06	203
4.2	1	> 3617	> 24379	2.21	5466	1.28	3461	0.23	730
4.3	0			> 23290	> 122793	1.64	4531	0.68	1991
`homcyclic`									
2	0	0.09	395	0.06	243	0.06	317	0.00	118
3	0	> 2580	> 14973	2.06	6558	0.14	545	0.01	169
4	1			> 48541	> 228118	1.88	4625	0.10	280
`infcyclic`									
2	-1	0.05	311	0.05	195	0.01	243	0.05	105
3	-1	> 11537	> 47863	1.15	2883	0.13	448	0.04	149
4	0			> 105057	> 494245	0.70	2165	0.13	481
5	0					65	1834	4	1507
6	1					1075	2948	114	2424
7	1					> 174167	> 16508	85204	21360

Computing the Dimension

The second task is to determine the exact dimension of the given variety. The correct test sequences (see §2.1) are mimicked by 10 points picked up at random with integer coordinates in $\{0, \ldots, 255\}$. The dimension computed is indicated in the second column of Table 7.

Comment on Table 7

First, we comment on the difference between the strategies `BDE` and `ED`. In the `BDE` method, it is necessary to construct the matrix of the linear application Q with polynomial entries. The application of Berkowitz's algorithm gives the determinant of the matrix Q^tQ in the form of a DAG, which has to be written down completely before proceeding to its evaluation. The big space requirement for this step prevents this method to work successfully on the bigger systems (lines 5, 6 of Table 7). This is due to the size of the matrices involved, which is indicated in Tables 8 and 9. The `ED` strategy improves upon the `BDE` strategy by first specializing the variables $x_0, \ldots, x_m$ with some integer values, and then treating the resulting matrix Q. Again interferes the size of the matrices (Tables 8 and 9), but this time over $\mathbb{Z}$. This step reduces the problem to the efficient computation of a determinant of an integer matrix. As illustrated in §2.3, this can be done by applying the Maple `det` function.

Next, the comparison between the `ED` and `EG` methods shows the effect of two different ways of implementing a Nullstellensatz function. In the `EG` method the matrix Q is not built at all, and the determinant is replaced by a Gröbner basis computation handled by Maple's `grobner` function. Indeed, performing a Buchberger algorithm on the $f_i^{(m)}$ (see

Table 8. Matrix size for `sac`

n	Matrix Size
sac.2.1	[10, 6]
sac.3.1	[10, 6]
sac.3.2	[35, 30]
sac.4.1	[10, 6]
sac.4.2	[35, 30]
sac.4.3	[126, 140]

Table 9. Matrix size for `infcyclic`

n	Matrix Size
3	[15, 19]
4	[36, 74]
5	[364, 875]
6	[969, 3161]
7	[14950, 54152]
8	[40920, 184206]
9	[101270, 548022]

Table 10. `infcyclic 8 EG-gb`

Dimension	Time per point tested
7	0.03
6	0.05
5	0.02
4	0.02, 0.07
3	2.66, 5.1
2	2660, 6400, 5700

Table 11. `homcyclic 8 EG-gb`

Dimension	Time per point tested
8	0.1
7	0.3, 0.2
6	0.02, 0.4
5	1.4
4	3.5, 5.3
3	191, 1057
2	> 50400

§3.2) can be seen as an efficient way of calculating a Gauss pivot on Q. This explains the better results for the `EG` method.

Finally, the comparison between the `EG` and the `g` methods seems to be in favor of the `g` method. What happens is that this strategy reduces the computation of the dimension for the system in n variables to k well-behaved Gröbner bases computations on systems in $n-d-1$ variables where d is the dimension and k is the number of test points. Here, well-behaved means that the complexity is bounded by that of a Gaussian elimination on the Q matrix. Thus our method performs best when the dimension is large.

Comment on Tables 8 and 9

Tables 8 and 9 give the size of the matrix Q which appears in the last step of the algorithms `BDE` and `ED`, the certification of the dimension. These matrices grow exponentially with the binomial coefficients depending on the codimension and the degrees of the polynomials involved. This growth is reflected by the time taken by any strategy (even `EG`). Since for small codimension this growth is still moderate, our approach is very efficient compared to a direct Gröbner basis computation on varieties of small codimension, even for systems in many variables.

Comment on Table 10 and 11

These tables illustrate the point made in §2.1 about the probabilistic nature of a zero-test with correct test sequences. Our algorithm is very fast for bounding the dimension of the variety and takes then some more time (corresponding to the certification step) to actually verify it (with bounded error probability). This is reflected by the time necessary for testing the chosen points.

4. Conclusions

The theoretical results on the polynomial complexity obtained by a straight-line program approach to the computation of the dimension are used here as a guide to an efficient implementation based on evaluation rather than rewriting.

When trying to certify the dimension, the a priori size of the matrices (and the DAG's) involved in the pure SLP strategy (BDE) prevents it from defeating rewriting techniques regularly, especially when they are implemented in an incremental way.

In the EG strategy which gives the best results, these straight-line programs themselves are never actually produced. However, the evaluations that are performed have an execution time which is bounded by the length that the corresponding straight-line program would have if written down. Thus we preserve the polynomial complexity while gaining speed and memory.

In situations of low codimension, our implementation gives an upper bound for the dimension and, with an extremely low (uniform) probability of failure, certifies that this bound is sharp much more quickly than the implementation of Gröbner basis used. In this sense it provides a very important speed up.

References

Balcázar, J. L., Díaz, J., Gabarró, J. (1995). *Structural complexity. I.* Texts in Theoretical Computer Science. An EATCS Series. Springer-Verlag, Berlin, second edition.

Berkowitz, S. J. (1984). On computing the determinant in small parallel time using a small number of processors. *Information Processing Letters*, 18(3):147–150.

Björck, G. (1990). Functions of modulus 1 on Z_n whose Fourier transforms have constant modulus, and "cyclic n-roots". In *Recent advances in Fourier analysis and its applications (Il Ciocco, 1989)*, volume 315 of *NATO Advance Science Institutes Series C: Mathematical and Physical Sciences*, pages 131–140. Kluwer Academic Publishers, Dordrecht.

Char, B. W., Geddes, K. O., Gonnet, G. H., Leong, B. L., Monagan, M. B., Watt, S. M. (1991). *Maple V Library Reference Manual*. Springer-Verlag.

Cox, D., Little, J., O'Shea, D. (1997). *Ideals, varieties, and algorithms*. Undergraduate Texts in Mathematics. Springer-Verlag, New York, second edition. An introduction to computational algebraic geometry and commutative algebra.

Eisenbud, D. (1995). *Commutative algebra*, volume 150 of *Graduate Texts in Mathematics*. Springer-Verlag, New York. With a view toward algebraic geometry.

Faugère, J.-C. (1995). *GB Reference Manual*. LITP. `http://posso.ibp.fr/GB.html`.

Faugère, J.-C. (1997). Gb: State of gb + tutorial. LITP.

Fitchas, N., Giusti, M., Smietanski, F. (1995). Sur la complexité du théorème des zéros. In *Approximation and optimization in the Caribbean, II (Havana, 1993)*, volume 8 of *Approximation and Optimization*, pages 274–329. Peter Lang Verlag, Frankfurt am Main. With the collaboration of Joos Heintz, Luis Miguel Pardo, Juan Sabia and Pablo Solernó.

Giusti, M. (1988). Combinatorial dimension theory of algebraic varieties. *Journal of Symbolic Computation*, 6(2-3):249–265. Computational aspects of commutative algebra.

Giusti, M., Heintz, J. (1993). La détermination des points isolés et de la dimension d'une variété algébrique peut se faire en temps polynomial. In Eisenbud, D., Robbiano, L., editors, *Computational algebraic geometry and commutative algebra (Cortona, 1991)*, volume XXXIV of *Symposia Matematica*, pages 216–256. Cambridge University Press, Cambridge.

Greuel, G.-M., Pfister, G., Schönemann, H. (1997). *Singular.* Universität Kaiserslautern, Germany. `http://www.mathematik.uni-kl.de/wwwagag/`.

Hearn, A. C. (1987). *Reduce user's manual, Version 3.3.* The RAND Corporation.

Heintz, J. (1989). On the computational complexity of polynomials and bilinear mappings. A survey. In *Applied algebra, algebraic algorithms and error-correcting codes (Menorca, 1987)*, volume 356 of *Lecture Notes in Computer Science*, pages 269–300. Springer, Berlin.

Heintz, J., Schnorr, C. P. (1982). Testing polynomials which are easy to compute. In *Logic and Algorithmic (Zürich, 1980)*, volume 30 of *Monographie de l'Enseignement Mathématique*, pages 237–254.

Lecerf, G., Schost, E. (1997). *Maple Package: GB link.* Laboratoire GAGE, École polytechnique, Palaiseau, France. `ftp://medicis.polytechnique.fr/pub/tera/soft/gblink`.

Mehlhorn, K., Näher, S., Uhrig, C. (1997). *Library for Efficient Datastructures and Algorithms.* Max Planck Institute for Computer Science, Saarbrücken. `http://www.mpi-sb.mpg.de/LEDA/leda.html`.

Mulmuley, K. (1987). A fast parallel algorithm to compute the rank of a matrix over an arbitrary field. *Combinatorica*, 7(1):101–104.

Pardo, L. M. (1995). How lower and upper complexity bounds meet in elimination theory. In Cohen, G., Giusti, H., Mora, T., editors, *Applied algebra, algebraic algorithms and error-correcting codes (Paris, 1995)*, volume 948 of *Lecture Notes in Computer Science*, pages 33–69. Springer, Berlin.

Stillman, M., Bayer, D. (1996). *Macaulay 2 User Manual.* `http://www.math.uiuc.edu/Macaulay2`.

Stoß, H.-J. (1989). On the representation of rational functions of bounded complexity. *Theoretical Computer Science*, 64(1):1–13.

Strassen, V. (1972). Berechnung und Programm. I, II. *Acta Informatica*, 1(4):320–355; *ibid.* 2(1), 64–79 (1973).

von zur Gathen, J. (1986). Parallel arithmetic computations: a survey. In *Mathematical foundations of computer science, 1986 (Bratislava, 1986)*, volume 233 of *Lecture Notes in Computer Science*, pages 93–112, Berlin. Springer.

Watt, S., Broadberry, P. A., Dooley, S. S., Iglio, P., Morrison, S. C., Steinbach, J. M., Stutor, R. S. (1995). *Axiom Library Compiler.* NAG.