Journal of Symbolic Computation 44 (2009) 222–231

Contents lists available at ScienceDirect

Journal of Symbolic Computation

journal homepage: www.elsevier.com/locate/jsc

Thirty years of Polynomial System Solving, and now?

Daniel Lazard

UPMC Univ Paris 06, LIP6, F-75005, Paris, France
INRIA Paris-Rocquencourt, SALSA project team, F-78153 Le Chesnay, France
CNRS, LIP6, F-75005, Paris, France

ARTICLE INFO

Article history:
Received 7 March 2008
Accepted 7 March 2008
Available online 25 September 2008

Keywords:
Polynomial system solving
Effective algebraic geometry
Cylindrical algebraic decomposition
Topology of semi-algebraic sets
Connected components
Parametric system
Algebraic optimization
Radical equidimensional decomposition

ABSTRACT

In this introductory paper to the special issue, I describe first my personal view of the history of *Polynomial System Solving* during my career. Then I describe the main challenges which are now opened by the availability of efficient zero-dimensional solvers.

1. Introduction

This special issue of *Journal of Symbolic Computation* follows a meeting held on December 2004 to celebrate my (administrative) retirement. This meeting and this issue are devoted to *Polynomial System Solving*, a subject on which I have been working for almost thirty years.

My retirement almost coincides with the availability in MAPLE 11 of the first algebraic solver (function `RootFinding[Isolate]`) which is able to solve automatically any medium sized zero-dimensional system (i.e. up to approximately hundred complex solutions). This is a major achievement of the domain which should change (and has already changed) the main directions of research in the field.

In this introductory paper, I will first try to explain why such an apparently narrow subject is worthy of a so long effort. To do this, I consider first the meaning of solving, which is less clear than it would seem and evolves with scientific progress. Then I will present my personal view of the history of the subject during my career and of the evolution of the main trends. As I am mainly concerned with the genesis of the above mentioned solver, this brief history is focused on zero-dimensional solving and Gröbner bases, omitting many aspects of the field which have a great interest in other contexts.

E-mail address: Daniel.Lazard@lip6.fr.

0747-7171/$ – see front matter
doi:10.1016/j.jsc.2008.03.004

Finally, I will expound my view of the main problems which have to be solved now and which are challenges for future research.

2. What is solving?

As with many informal questions, the answer to the question *what is solving?* is not evident. Moreover, its answer varies with the progress of solving methods. In fact this question may be split into several questions for which I give separate answers.

What is Polynomial System Solving? Usually, a polynomial system is a finite set $f_1 = 0, \ldots, f_k = 0$ of equations whose first members are multivariate polynomials with coefficients in some field K, which is, in practice, the field of the rationals $\mathbb{Q}$ or a finite field. By *solving*, we mean to find *all* the solutions in some algebraically closed extension $\overline{K}$ of K or in the field of the real numbers. In this paper, we will focus on real or complex solutions, which are especially important in applications. Note that we are not considering here the Diophantine equations for which only solutions inside $\mathbb{Q}$ are looked for. We exclude also iterative methods, such as Newton's, which eventually produce some solutions but never prove that all have been found. In addition in the real case most systems contain also inequalities together with the equations.

What does means "find the solutions"? The answer to this question is very different depending on whether the number of solutions is finite or not. Thus this is the first thing to decide. A system is called *zero-dimensional* if the number of its solutions is finite in any algebraically closed field, *positive dimensional* if not. Thus solving a zero-dimensional system consists in producing the list of all solutions. But this introduces another question: *How the solutions are represented?* or in modern computer science language: *What should be the output specification of the algorithms?* There are several ways to represent the solutions; however, if K is the field $\mathbb{Q}$ of the rationals, to be useful in practice the representation of the solutions should contain a real or a complex approximation of the solutions (or should allow it to be computed easily). In the case of the algebraic closure of a finite field, where approximations are not available, another representation is needed. Note that this question of the output specification is the modern form of the long standing debate on solving univariate equations either by radicals or by approximations. It also underlies the problem of constructions by ruler and compass (quadrature of the circle, duplication of the cube and trisection of an angle), which is now known to be equivalent to solving an equation by square roots.

What does "solving" mean in the case of a positive dimensional system? Clearly, it is impossible to produce a list of all the solutions. Thus in this case, *solving* consists in producing a description of the set of the solutions which allows one to extract the desired information easily. It follows that the output specification should be a compromise between what is computable and what is useful. This question is therefore very hard and a complete answer is probably impossible.

What is the relationship between "Polynomial System Solving" and "Algebraic Geometry"? Roughly speaking, Algebraic Geometry is the *study* of the solution sets of a polynomial system. Thus, the mathematical foundation of both fields is the same. However, a difference of the point of view and of objectives introduces a great difference which may be illustrated by two examples: It is a basic theorem of Algebraic Geometry that any algebraic variety (i.e. any solution set) is a finite union of irreducible ones. Thus Algebraic Geometry starts from this decomposition and is mainly devoted to the study of irreducible varieties. On the other hand, when solving a system, it is *a priori* unknown if the solution set is irreducible, and its decomposition in irreducible components is a difficult task, not yet well solved; therefore this decomposition should be avoided as far as possible. The second example lies in the zero-dimensional systems which are viewed as essentially trivial in Algebraic Geometry while they are fundamental and difficult to solve when one is interested in effective solving.

The importance of these questions seems to be frequently undervalued, especially the need for clear specification of the problem to be solved as well as of the algorithms developed to solve it. However, when I began to work on the subject, at the end of the seventies, they were completely ignored.

3. Solving zero-dimensional systems through Gröbner bases

In this section, I sketch my personal view of the history of *polynomial system solving*. I concentrate on practical solving and on long term trends, without trying to be complete. Thus I omit almost all results on complexity as well as the algorithms and methods, which have not yet shown to be efficient in practice for zero-dimensional systems or have been outclassed by more recent algorithms.

During the twentieth century, the main practical problem was the zero-dimensional case (i.e. the systems with a finite number of solutions in an algebraically closed field). The result of these thirty years of experience is that the best algorithms for this case are all related to Gröbner bases.[1] Therefore I focus on Gröbner bases and related algorithms, omitting other methods which, with the present state of the art, are less efficient than Gröbner bases for zero-dimensional systems.

The first time I was faced with Polynomial System Solving was around 1970: In order to see how computers could be useful in Mathematics, I was testing the so called Serre's problem,[2] and this led me to see if there exists a solution of the equation

$$\begin{vmatrix} 1+Y+Z+XZ & A_1 & A_4 \\ Z-XY & 1+A_2 & A_5 \\ Y^2+YZ+Z^2 & A_3 & 1+A_6 \end{vmatrix} = 1 \tag{1}$$

where the unknowns A_i are polynomials in $\mathbb{Q}[X, Y, Z]$ without a constant term (here the equality means an equality of polynomials). Without a better method, I supposed that the A_i had a degree not higher than 2, which leads to a system of 81 quadratic equations with the 54 coefficients of the A_i as unknowns. At that time (and probably today also), the only way to solve such a system was to compute, with an optimization program, some local minimums of the sum of the square of the equations, hoping eventually to find zero. To do this, one has first the write down (into a routine of evaluation) this sum of squares and its gradient. Unable to do this by hand, I learnt of the existence of an ancestor of the computer algebra systems called FORMAC which was able to do this. Finally, after trying various algorithms, programs and starting points, I found the solution[3]

$$\begin{aligned} &A_1 = X - Y - Z - XZ, \quad A_2 = XY - Z, \quad A_3 = Y + Z + XY - Y^2 - YZ - Z^2, \\ &A_4 = Y + Z + XZ, \quad A_5 = X + Z - XY, \quad A_6 = XZ - Y + Y^2 + YZ + Z^2. \end{aligned}$$

This did not shed any light on Serre's problem, but showed me the lack of available mathematical algorithms and the large amount of work which is needed to transform a theoretical algorithm to a working implementation.[4]

This is this experiment which led me to devote my research to the algorithms of Computer Algebra. The first algorithms I considered were the resolution of linear systems with polynomials as coefficients and unknowns. Eq. (1) is an example of such a system if the polynomials A_1, A_2 and A_3 are given. This led me to the paper Lazard (1977), and, soon after, I noticed that the methods of this paper might be used for Polynomial System Solving, which was clearly the most fundamental problem of Computer Algebra which was unsolved in practice: At that time, the algorithms for polynomial factorization and polynomial GCD (essentially due to Collins and his school) allowed one to solve problems far beyond the possibilities of hand computation, while exact algorithms for Polynomial System Solving were limited to very simple academic problems in two or three variables.

---

[1] In this paper, I am only concerned with *Algebraic solving*, and thus I omit all approximate methods, like the homotopy approach of Vershelde.

[2] The problem was: *are all projective modules over a polynomial ring free?* This has been proved true independently by Quillen (1976) and Suslin (1976).

[3] The true output was floating point numbers, which were very close to 0, 1 or −1. Replacing them by integers, it was easy to verify that the obtained solution was correct. Note that this way of getting an integer solution is a strong indication that the solution is probably unique.

[4] At that time, very few mathematicians knew any programming. So it was much later that I heard the following sentence which is common for mathematicians who are interested in theoretical computation but have no practical experiment of programming: "It is trivial, it is only implementation!".

In fact, in the seventies, there were already efficient numerical algorithms for finding solutions of polynomial systems like the one above, but there were no way to know if all the solutions had been found (the above example of 81 equations with 54 unknowns shows the capabilities already reached by numerical methods). On the other hand, for algebraically solving a polynomial system, the algorithms which were implemented consisted essentially in iterating resultant computations. This eventually produces a univariate polynomial, some of whose roots are the first coordinates of the solutions. It thus remains to decide which roots correspond to solutions and to compute the other coordinates. This approach is not practical for two reasons: First, each time a variable is eliminated, the degree of the polynomial is doubled, leading to a univariate polynomial of doubly exponential degree. Second, the computation of the other coordinates needs to compute the GCD of and to solve polynomials with algebraic coefficients, which is not an easy task, even in the case of two variables.

At the meeting EUROCAM'79 (Ng, 1979) (an ancestor of ISSAC held in Marseilles), I presented the first solving algorithm (based on multivariate resultants) which has a complexity which is simply exponential in the number of the variables (Lazard, 1981). At the same meeting, Bruno Buchberger presented the Gröbner bases and his two criteria to eliminate unnecessary critical pairs. The fact the Gröbner basis algorithm is a major tool for polynomial system solving was already known at that time (Trinks, 1978), but the relationship between the two methods was discovered only several years later (Lazard, 1983).

At the beginning of the eighties, none of the solving methods was efficient enough to solve non trivial problems (i.e. problems not solvable or difficult to solve by hand computation). Thus most of the research work was devoted to the theoretical improvement of the algorithms and to the computation of their complexity. Also, several alternative methods for solving were proposed, the main one being the Ritt–Wu Wentsün approach through characteristic sets. It appeared that, as any solving algorithm has a complexity which is exponential in the number of variables (because the number of solutions has this exponential behavior), the worst case complexity is usually doubly exponential (however it is not known for the Ritt–Wu Wentsün method) and that a single exponential complexity may be reached in most (sufficiently regular) cases. However, practical implementations were unable to reach this single exponential complexity, mainly because of the growth of the intermediate expression.

This had the consequence that, from a practical point of view, there were only experimental implementations, which were rarely able to solve problems not accessible by hand. When such a problem was solved, the algebraic methods provided only the number of solutions, proving that the set of solutions which were found by other methods were complete. This is especially the case for the system of equations known as Cyclic(n) which were proved to have respectively 50 solutions for $n = 5$ in 1987 (Davenport, 1987), 156 for $n = 6$ soon after (apparently not published) and 924 for $n = 7$ in 1991 (Backelin and Fröberg, 1991).

Nevertheless many important algorithmic progresses appeared during this period or at its end, at the beginning of the nineties. I may cite the following ones. The recognition that the Gröbner basis for the reverse degree lexicographical ordering is easier to compute than the basis for the purely lexicographical ordering[5] and that any ordering allows one to verify that a system is zero-dimensional and to count its number of solutions. The "sugar" strategy (Giovini et al., 1991) usually dramatically improves Buchberger's algorithm. Linear Algebra allows one to compute efficiently the basis for the lexicographical ordering (the best suited for solving) from the basis for the degree ordering (the easiest to compute); this algorithm, now known as the FGLM algorithm (Faugère et al., 1993), appeared in Journal of Symbolic Computation only in 1993, but it had been known since 1989. Finally, it is easy to deduce from a lexicographical Gröbner basis a decomposition of the set of zeros into a finite set of triangular systems, allowing one to compute the solutions without GCD of polynomials with algebraic coefficients (Lazard, 1992).

Thus, at the beginning of the nineties, the best strategy to solve zero-dimensional systems was the following one: compute the Gröbner basis for the degree reverse lexicographical ordering; deduce

[5] This is implicit in Lazard (1983), but it took some time to be well recognized by the community.

by the FGLM algorithm the Gröbner basis for the lexicographical ordering; decompose the result into triangular sets; solve recursively the univariate polynomials thus obtained.

This strategy still remains the most efficient for systems over finite fields which appear in Cryptography, but has been outclassed in characteristic zero by the RUR algorithm which shortcuts all these steps but the first one (Rouillier, 1999).

In the years following 1991, few theoretical results and new algorithms related to Gröbner bases have appeared. The reason seems to be that most researchers work was devoted to the effort of implementation: It is in this period that all specialized software was developed (Macaulay2, Singular, Cocoa, GB) and all general computer algebra systems began to include Gröbner base packages in their distribution.

Although it is rather easy to implement Buchberger algorithm crudely, the huge amount of data generated and the length of the computation makes it very difficult to obtain an efficient implementation. In particular, when working with rational coefficients the efficiency of the arithmetic is critical. At that time, the GMP package for multiple precision arithmetic was not yet available and therefore most of the specialized packages allowed only computation on finite fields. On the other hand, the packages of the general systems were very inefficient because of their data structure for polynomials, which were not well suited for Gröbner basis algorithm. Thus, in 1993, the same Gröbner basis computation took 3h30 with Maple, 4' with Axiom and 3'' with GB (Lazard, 1993); as it was a computation with rational coefficients, it was impossible with the other available specialized packages.

When Gröbner basis packages became available, it became clear to me that computing a Gröbner basis was not really solving: A Gröbner basis is usually a very large data set which is not, by itself, the solution of any problem. Therefore other software is needed to deduce the solutions from the Gröbner basis, which may only work if the Gröbner basis has been computed over the rational numbers.

It seems that the importance of viewing solving as a post-processing of a Gröbner basis computation has been undervalued by most teams developing Gröbner basis implementations: they do not propose any solving function, or if they do, few efforts have been devoted to it.

On the contrary, within my team a competition began between J.-C. Faugère for computing Gröbner bases faster and F. Rouillier for computing the solutions from the huge Gröbner bases provided by Faugère. This competition resulted in several new algorithms (Faugère, 1999, 2002; Rouillier, 1999; Rouillier and Zimmermann, 2003) and in a set of software[6] which is now included in Maple.

This set of software allows one to compute routinely and in a certified way the solutions of zero-dimensional systems having a few hundreds of solutions. Thus I consider that the *problem of solving zero-dimensional systems is now well solved*, even if a lot of work is yet needed to get better efficiency, because, frequently, the systems encountered in applications greatly exceed this bound.

4. Positive dimension: What is the challenge?

As already said, we now have rather efficient software for solving zero-dimensional polynomial systems which have a finite number of solutions.

Gröbner bases, as well as the other basic tools for dealing with polynomial systems (essentially Collins Cylindrical Algebraic Decomposition and triangular sets) allow one to get some information on systems which are not zero-dimensional. The information which may be obtained from these three different approaches is not the same. However, in many cases, the same information may be obtained in different ways. Thus the algorithm designer is faced with three problems. First, he has to decide which information is useful in many applications, or, in other words, to choose the output specification of the algorithms. Second, he has to choose the most efficient method, among those which allow one to fulfill these specifications. Finally he has to design an algorithm and implement it.

Clearly these problems are not completely independent. For example, if the specification is too general, the complexity may be too high to allow practical computations whichever algorithm is

[6] SALSA Software: http://fgbrs.lip6.fr/salsa/Software/.

chosen. One of the reasons for this is that general algorithms compute usually much more than what is really needed.

Although these problems are not independent I will examine them separately to show that all are important challenges for Computer Algebra.

4.1. Challenge 1: What is to be computed?

This question is almost the same as the above question *"What does mean solving?"*, but it becomes clearer since we have efficient software for zero-dimensional systems.

In fact, there are more and more people, coming from various areas, which ask questions relative to polynomial system solving. Usually these problems are not formulated in terms of polynomials systems. Even when they are, the question which is asked is rarely the true question for which an answer is needed. Frequently, some "simpler" question is asked and this simplification may make the problem unsolvable; linearization is such a simplification, which works for numerical computation but not for algebraic computation. In other cases people do not know exactly what they want. Here is an example: A researcher in computational geometry got the condition for degeneracy of some geometrical problem as the vanishing of a polynomial in six variables which need several pages to be printed. His question was simply *"What can I do with this polynomial?"*.

Here are four typical questions which have been posed to me or to other members of my team. *"When a serial robot may change posture without moving its end effector?"* (Corvez and Rouillier, 2003), *"Is a given polynomial map injective in the domain where its definition is meaningful?"* (Lazard, 2004), *"Given a polynomial multivariate function depending on some parameters, for which values of the parameters is there no other local minimum than the global one?"* (Gu et al., 2007), *"Has a given hypersurface other real points than its singular points?"* (Everett et al., 2007).

These questions are not difficult to transform in terms of quantified formulas involving polynomial equations and inequalities. The quantifiers could be eliminated by Collins Cylindrical Algebraic decomposition. The resulting formulas are systems of equations and inequalities. Thus it easy to find a solving process, which is unfortunately far from being able to compute the solutions effectively.

After having worked some time on these problems and solved some of them by ad hoc methods, we have realized that, for the first three problems, these ad hoc methods consisted in reducing them to the same general problem: Given a system of equations and inequalities depending on parameters, to decompose the space of parameters in subsets where the number of solutions and their behavior are constant. Then the solution of the initial problem may be obtained by solving a zero-dimensional system for a sample point in each of these subsets.

Thus the answer to the question *What is to be computed?* is far from being easy: it needs work with specialists of various areas to understand which of their problems may be solved by polynomial systems solving and how to solve them. Then the solutions which have been obtained have to be abstracted into problem specifications of general interest. Finally, efficient algorithms for these specifications have to be designed.

For the moment, I am able to enumerate four classes of such problems of general interest. This list is certainly incomplete and new problems will certainly appear in the near future. Nevertheless, all these problems are incompletely specified and each one is a long term challenge for young researchers.

Parametric systems. *Given a system of equations and inequalities depending on parameters, the problem consists in decomposing the space of the parameters into open*[7] *subsets where the topology and the algebraic structure of the set of solutions is constant*. The number of open subsets should be close to the optimal to be useful in the applications (and also to allow efficient computation). This problem has been solved when the system is zero-dimensional for almost all values of the parameters (Lazard and Rouillier, 2007). The obstruction to a

[7] This means that the space of the parameters is the union of the closure of these open subsets. What happens on the frontier of these open subsets is certainly much more difficult to study and is usually of less interest. Nevertheless, one may decide what happens on this frontier by adding the equations of the frontier to the initial equations and running the algorithm again.

generalization is of a mathematical nature: we do not have any characterization of the closed subset of the parameter space where the topology of the set of solutions changes.

Connected components of a semi-algebraic set. *Given a system of equations and inequalities, the problem consists in producing exactly one point in each connected component of the set of the solutions*. There exist algorithms for this problem but they are too inefficient to have been implemented. There exist implementations which produce at least one point in each connected component.[8] They do not allow one to count the connected components but allow one to test if the set of solutions is empty. In particular they allow one to test if a multivariate polynomial is always non negative (the set defined by $f < 0$ is empty) and if the hypersurface which it defines has only singular real points (in this case one of the sets defined by $f < 0$ or $f > 0$ is empty) (Safey El Din, 2007).

Optimization. *The problem is to find the minimum of a polynomial under polynomial constraints.* Many instances of the problem reduce to a single zero-dimensional system: the minimum is obtained at a point where the gradient of the polynomial is orthogonal to the variety of the constraints. Things are more difficult if the hypersurface defined by the polynomial has non isolated singularities (in this case the system is not zero-dimensional) or if the minimum is not attained (i.e. it is the limit of values of the polynomial for a sequence of points tending to infinity). The main challenge here is to classify the optimization problems where the algebraic methods are more efficient or more robust than the numerical methods.

Topology of semi-algebraic sets. This is a long term challenge which generalizes the above problem of computing the number of connected components. It is not even clear what should be the output of an algorithm which computes the topology of a semi-algebraic set.

Remark 1. The first and third challenges are quantifier elimination problems and may theoretically be solved by Collins CAD. The second challenge may also be solved by CAD if one is able to test cell adjacency. However CAD is too inefficient to solve most practical problems which may be reduced to these challenges.

Conversely, the two first challenges, if solved, allow one to eliminate quantifiers in purely existential formulas (it suffices to take the free variables as parameters). This shows that the class of quantifier elimination problems is much larger than the class of problems which may reduced to these challenges. Thus the complexity of these problems is certainly much lower than the general quantifier elimination problem.

The inefficiency of CAD to eliminate quantifiers in purely existential formulas is highlighted by the existence of problems solved by hand computation which are too difficult for automatic computation. This is the case for the following problem which was posed to us around 1999 by a Ph.D. student named Rozier. It consists in eliminating the quantifiers in

$$\exists(a,b,c,d)\ \ \forall x\ \ (P > 0\ \ \wedge\ \ c+d > 0\ \ \wedge\ \ d(a-1) > 0)$$

where $P = (x+c)(x^3-u) + (x-d)(bx+av)$. This is not a purely existential formula, but it is easy to eliminate the universal quantifier to get a purely existential formula. Although solved by hand (an exercise for the reader) this problem is still a challenge for automatic computation.

4.2. Challenge 2: How to compute?

When trying to design and implement algorithms to solve non zero-dimensional systems, one is faced with some contradiction in the state of the art: Most problems are of a geometric nature, while most basic tools are purely algebraic (i.e. they take the multiplicities into account although they are of little interest in most applications). Moreover most algebraic tools consider the multivariate polynomials as univariate polynomials with polynomials as coefficients; this hides the invariance of the geometric properties under the action of the linear group.

[8] See for example RAGLIB in http://fgbrs.lip6.fr/salsa/Software/.

This makes it difficult to design efficient algorithms because it is clear that the only way to get an efficient algorithm for a geometrical problem is to take into account the geometrical properties of the problem. When such an algorithm is designed, its implementation remains difficult because the basic tools of the existing software are not adapted to the need of the algorithm.

This suggests to me that a solution consists of defining a small number of well chosen geometrical primitives and splitting the job into two separate actions: on one hand constructing an efficient implementation of these primitives; on the other hand designing the geometrical algorithms in term of these primitives.

It seems to me that this program is feasible, because many recent geometrical algorithms may be rewritten in term of the primitives which are described below. This is especially the case for all known partial solutions of the above challenges.

It appears that a good choice for these general primitives is the following. This list is certainly not complete and new important primitives will certainly appear with the progress of the state of the art.

Inclusion between ideals. When the ideals are given by generators, this is easily tested by computing a Gröbner basis (for any monomial ordering, but the computation is faster with the degree reverse lexicographical ordering) of the largest ideal. In many cases, a test of inclusion between the varieties of zeros of the ideals would be better, but it is much more costly, and thus needs to be avoided as far as possible.

Dimension of the (complex) variety of the zeros of an ideal. This is easily deduced from the Gröbner basis of the ideal, for any monomial ordering.

Real zero-dimensional solving. This means, given a zero-dimensional system of equations, to isolate the real solutions in boxes of a small given maximal dimension. The best (and maybe unique) current implementation is that of SALSA Software.[9]

Projection/elimination. Given a set of variables $\{x_1, \ldots, x_n\}$ the projection of an algebraic set $V \subset \mathbb{R}^n$ onto the subspace $\mathbb{R}^k$ defined by the k first variables consists algebraically in the elimination of $x_{k+1}, \ldots, x_n$ in the ideal I defining V, i.e. in computing $I \cap \mathbb{Q}[x_1, \ldots, x_k]$ (if the field of the coefficients is $\mathbb{Q}$). A Gröbner basis of this ideal is the set of the polynomials independent of $x_{k+1}, \ldots, x_n$ in a Gröbner basis of I for a block ordering.

Localization/saturation. Given an algebraic set V defined by an ideal I and a hypersurface S defined by a polynomial f, the localization of V by S or the saturation of I by f consists in removing from V the irreducible components contained in S or, equivalently, in removing from I the primary components containing a power of f. This may be done by eliminating the new variable u in the ideal generated by $I \cup \langle 1 - uf \rangle$.

Radical equidimensional decomposition. This operation consists in decomposing an algebraic set into equidimensional algebraic sets defined by radical ideals, or, algebraically, in decomposing the radical of a given ideal as the intersection of a finite set of equidimensional ideals. It may be computed either as a partial output of a primary decomposition algorithm or by a specific algorithm like Kalkbrenner's one (Kalkbrener, 1993).

To rewrite the algorithms of Real Algebraic Geometry in terms of such primitives has many advantages. First the algorithms and their implementation become independent of the basic algebraic tools: if better algorithms, either involving or not involving Gröbner bases, are found for the primitives, one has only to change some links in the programs. Second, as these primitives are of a geometric nature, it is easier to not lose the geometric nature of a problem by using them in the solution; this may allow one to use the geometric properties of the problem to optimize the resolution. Finally, as the algorithms may be very close to the geometric nature of the problem, their complexity may be very close to the optimum, if the algorithms used for the primitive would be optimal.

Another advantage is that there is no need to use the arithmetic of the real algebraic numbers. This arithmetic is highly costly and is a reason for the inefficiency of Cylindrical decomposition when the number of variables is higher than three. This arithmetic is usually viewed as unavoidable, because, in Geometry, most points are defined by intersections and have thus algebraic coordinates.

[9] http://fgbrs.lip6.fr/salsa/Software/.

If a point is stored as a zero-dimensional system together with inequalities specifying which root is considered, the above primitives allow one to answer any question on the points (comparing coordinates, inclusion in a semialgebraic set, ...). Moreover, the output of zero-dimensional solving is a small box containing the point and thus a floating point approximation of the point. For stable questions, this allows one to use the efficiency of floating point computation.

The radical equidimensional decomposition is the only above primitive which may not be implemented by a single Gröbner basis computation. In practice it is rarely needed, but when it is needed, it is frequently a critical step, and an algorithm which is much more efficient than those which are available would be welcome.

The saturation is frequently a key primitive, as it allows to remove some degeneracies. It is frequent that the key point in designing algorithms in real algebraic geometry is to decide when and by which polynomial to saturate. A very simple example is the problem of implicitization.

This problem consists of, given a rational parametrization of a variety, finding its implicit equations. Let $x_i = f_i(t_1, \ldots, t_k)/f_0(t_1, \ldots, t_k)$ for $i = 1, \ldots, n$, the f_i being polynomials, be a parametric representation of an algebraic variety. The elementary idea to getting implicit equations of the variety is to eliminate the x_i between the polynomials $f_0x_i - f_i$. However this does not work directly if the f_i have a common zero (which is usually called a *base point*). Thus the standard algorithm to implicitize may be expressed in term of our primitives as: *saturate by f_0 the ideal generated by the polynomials $f_0x_i - f_i$; eliminate the x_i from the result.* If the first step is omitted and there are base points, the theoretical consequence is that one should obtain the null ideal, but, in practice, elimination is much more time consuming than with the correct algorithm and, frequently, the computation may not be finished.

This simple example shows the importance in algorithm design of choosing wisely when a saturation should be done; this may be needed for the correction of the algorithm, but, even when it is not necessary for the correction of the algorithm, this may dramatically change the efficiency.

On the other hand, the primitive *radical equidimensional decomposition* seems to be less frequently needed, maybe because the ideals coming from applications are usually prime (in the above example of the implicitization, the ideal obtained by the saturation step is always prime). However, when radical equidimensional decomposition is needed, this is frequently a blocking step.

Therefore, I find it important to obtain better radical equidimensional decomposition algorithms in order that this step does not remain a blocking one. This seems to be a short term challenge, because, each time I have had to compute such a decomposition in an application problem, I was able to get it efficiently by some *ad hoc* methods combining the specificity of the problem with the various existing methods (Kalkbrenner representation of the components by triangular chains, computation of these triangular chains by Gröbner bases and various tools used in the algorithms of primary decomposition).

5. Conclusion

In this informal paper, I have tried to sketch my personal history of *polynomial system solving* through *Gröbner bases*. As the subject is very wide, I have emphasized the aspect of *problem solving*, which seems to be the most important (at least for people outside the field of Symbolic Computation and for funding agencies).

My (administrative) retirement coincides with the availability of efficient software for solving zero-dimensional systems. This does not close the problem of *polynomial system solving*. On the contrary, this allows one to consider now the problem of *solving polynomial systems of positive dimension*. This opens up a number of challenges which may be classed in two categories, the first one consisting in defining and specifying the problems to solve. The second is of a more classical algorithmic nature. However, the existence of efficient software for zero-dimensional solving allows one to develop a geometrical algorithmic for geometrical problems.

Thus *polynomial system solving* will probably remain an active domain for a long time, possibly another period of thirty years.

References

Backelin, J., Fröberg, R., 1991. How we proved that there are exactly 924 cyclic 7-roots. In: ISSAC'91: Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation, pp. 103–111.

Corvez, S., Rouillier, F., 2003. Using computer algebra tools to classify serial manipulators. In: Winkler, F. (Ed.), actes de Automated Deduction in Geometry. In: Lecture Notes in Artificial Intelligence, vol. 2930. pp. 31–43.

Davenport, J.H., 1987. Looking at a set of equations. Tech. Rep., Bath Computer Science.

Everett, H., Lazard, D., Lazard, S., Safey El Din, M., 2007. The Voronoi diagram of three lines in $\mathbb{R}^3$. In: Proceedings of the Symposium on Computational Geometry (Gyeongju, South Korea). p. none.

Faugère, J.-C., Gianni, P., Lazard, D., Mora, F., 1993. Efficient computation of zero-dimensional Gröbner bases by change of ordering. Journal of Symbolic Computation 16 (4), 329–344.

Faugère, J.-C., 1999. A new efficient algorithm for computing Grobner bases (F4). Journal of Pure and Applied Algebra 139, 61–88.

Faugère, J.-C., 2002. A new efficient algorithm for computing Gröbner bases without reduction to zero (F5). In: Proceedings of the International Symposium on Symbolic and Algebraic Computation. ACM Press, pp. 75–83.

Giovini, A., Mora, F., Niesi, G., Robbiano, L., Traverso, C., 1991. One sugar cube, please or selection strategies in the Buchberger algorithm. In: ISSAC '91: Proceedings of the 1991 International Symposium on Symbolic and Algebraic Computation. ACM Press, pp. 49–54.

Gu, N., Lazard, D., Rouillier, F., Xiang, Y., 2007. Using computer algebra to certify the global convergence of a numerical optimization process. Mathematics in Computer Science 1 (2), 291–304.

Kalkbrener, M., 1993. A generalized Euclidean algorithm for computing triangular representations of algebraic varieties. Journal of Symbolic Computation 15 (2), 143–167.

Lazard, D., 1977. Algèbre linéaire sur $K[X_1, \ldots, X_n]$, et élimination. Bulletin de la Sociéte Mathématique de France 105 (2), 165–190.

Lazard, D., 1981. Résolution des systèmes d'équations algébriques. Theoretical Computer Science 15 (1), 77–110.

Lazard, D., 1983. Gröbner bases, Gaussian elimination and resolution of systems of algebraic equations. In: Computer algebra (London, 1983). In: Lecture Notes in Comput. Sci., vol. 162. Springer, Berlin, pp. 146–156.

Lazard, D., 1992. Solving zero-dimensional algebraic systems. Journal of Symbolic Computation 13 (2), 117–131.

Lazard, D., 1993. On the representation of rigid-body motions and its application to generalized platform manipulators. In: Angeles, J., Hommel, G., Kovács, P. (Eds.), Computational Kinematics. In: Solid Mechanics and its Applications, No. 29. Kluwer, pp. 175–181.

Lazard, D., 2004. Injectivity of real rational mappings: The case of a mixture of two Gaussian laws. Mathematics in Computers in Simulation 67 (1), 67–84.

Lazard, D., Rouillier, F., 2007. Solving parametric polynomial systems. Journal of Symbolic Computation 42, 636–667.

Ng, E.W. (Ed.), 1979. Proceedings of the EUROSAM 79, Symposium on Symbolic and Algebraic Manipulation. Marseille, June 26–28, 1979. In: Lecture Notes in Computer Science, vol. 72. Springer, Berlin, Heidelberg, New York, London, UK.

Quillen, D., 1976. Projective modules over polynomial rings. Inventiones Mathematicae 36, 167–171.

Rouillier, F., 1999. Solving zero-dimensional systems through the rational univariate representation. Journal of Applicable Algebra in Engineering, Communication and Computing 9 (5), 433–461.

Rouillier, F., Zimmermann, P., 2003. Efficient isolation of polynomial real roots. Journal of Computational and Applied Mathematics 162 (1), 33–50.

Safey El Din, M., 2007. Testing sign conditions on a multivariate polynomial and applications. Mathematics in Computer Science 1 (1), 177–207.

Suslin, A.A., 1976. Projective modules over polynomial rings are free. Rossiĭskaya Akademiya Nauk. Doklady Akademic Nauk. SSSR 229 (5), 1063–1066 (in Russian).

Trinks, W., 1978. Über B. Buchbergers Verfahren, Systeme algebraischer Gleichungen zu lösen. Journal of Number Theory 10 (4), 475–488.