On the Complexity of Real Solving Bivariate Systems*

[Extended abstract]

Dimitrios I. Diochnos
National University of Athens
Athens, Hellas
d.diochnos(at)di.uoa.gr

Ioannis Z. Emiris
National University of Athens
Athens, Hellas
emiris(at)di.uoa.gr

Elias P. Tsigaridas
LORIA-INRIA Lorraine
Nancy, France
elias.tsigaridas(at)loria.fr

ABSTRACT

We consider exact real solving of well-constrained, bivariate systems of relatively sparse polynomials. The main problem is to compute all common real roots in isolating interval representation, and to determine their intersection multiplicities. We present three algorithms and analyze their asymptotic complexity, obtaining bounds of $\tilde{O}_B(\sqrt{N^{14}})$ for the purely projection-based method, and $\tilde{O}_B(\sqrt{N^{12}})$ for two subresultants-based methods: these ignore polylogarithmic factors, and $N$ bounds the degree and the bitsize of the polynomials. The previous record bound was $\tilde{O}_B(\sqrt{N^{14}})$.

Our main tool is signed subresultant sequences, extended to several variables by binary segmentation. We exploit advances on the complexity of univariate root isolation, and extend them to multipoint sign evaluation, sign evaluation of bivariate polynomials over two algebraic numbers, and real root counting over an extension field. Our algorithms apply to the problem of simultaneous inequalities; they also compute the topology of real plane algebraic curves in $\tilde{O}_B(N^{12})$, whereas the previous bound was $\tilde{O}_B(N^{14})$.

All algorithms have been implemented in MAPLE, in conjunction with numeric filtering. We compare them against FGb/R5 and SYNAPS; we also consider MAPLE libraries INSULATE and TOP, which compute curve topology. Our software is among the most robust, and its runtimes are within a small constant factor, with respect to the C/C++ libraries.

Categories and Subject Descriptors: F.2.0 [Analysis of Algorithms and Problem Complexity]; General; G.4 [Mathematical software]: Algorithm design and analysis;

General Terms: Algorithms, Experimentation, Theory

Keywords: polynomial system, real algebraic number, real solving, topology of real algebraic curve, maple

*All authors acknowledge partial financial support by PET-Open European Project ACS (Algorithms for complex shapes). The third author started work on this project while at University of Athens and INRIA Sophia-Antipolis. The third author is also partially supported by ARC ARCADIA.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

Copyright 2007 ACM 978-1-59593-743-8/07/0007 ...$5.00.

1. INTRODUCTION

The problem of well-constrained polynomial system solving is fundamental. However, most of the algorithms treat the general case or consider solutions over an algebraically closed field. We focus on real solving of bivariate polynomials, in order to provide precise complexity bounds and study different algorithms in practice. We expect to obtain faster algorithms than in the general case. This is important in several applications ranging from nonlinear computational geometry to real quantifier elimination. We propose relatively prime polynomials for simplicity, but this hypothesis is not restrictive. A question of independent interest is to compute the topology of a plane real algebraic curve.

Our algorithms isolate all common real roots inside non-overlapping rational rectangles, and output them as pairs of algebraic numbers; they also determine the intersection multiplicity per root. In this paper, $\tilde{O}_B$ means bit complexity and $\tilde{O}_B$ means that we are ignoring polylogarithmic factors. We derive a bound of $\tilde{O}_B(N^{12})$, whereas the previous record bound was $\tilde{O}_B(N^{14})$, see also [2], derived from the closely related problem of computing the topology of real plane algebraic curves, where $N$ bounds the degree and the bitsize of the input polynomials. This approach depends on Thom’s encoding. We choose the isolating interval representation, since it is more intuitive, it is used in applications, and demonstrate that it supports as efficient algorithms as other representation. In [2] it is stated that “isolating intervals provide worst [sic] bounds”. Moreover, it is widely believed that isolating intervals do not produce good theoretical results. Our work suggests that isolating intervals should be re-evaluated.

Our main tool is signed subresultant sequences (closely related to Sturm-Habicht sequences), extended to several variables by the technique of binary segmentation. We exploit the recent advances on univariate root isolation, which reduced complexity by 1-3 orders of magnitude to $\tilde{O}_B(N^6)$ [11] [13]. This brought complexity closer to $\tilde{O}_B(N^4)$, which is achieved by numerical methods [21].

In [15], $2 \times 2$ systems are solved and the multiplicities computed under the assumption that a generic shear has been obtained, based on [20]. In [16], $2 \times 2$ systems of bounded degree were studied, obtained as projections of the arrangement of 3D quadrics. This algorithm is a precursor of ours, see also [11], except that matching and multiplicity computation was simpler. In [22], a subdivision algorithm is proposed, exploiting the properties of the Bernstein basis, with unknown bit complexity, and arithmetic complexity based on the characteristics of the graphs of the polynomials. For
other approaches based on multivariate Sturm sequences the reader may refer to e.g. \[20\] [20].

Determining the topology of a real algebraic plane curve is a closely related problem. The best bound is \(O_B(N^{12})\) \[12\]. In \[17\] three projections are used; this is implemented in \textsc{insulate}, with which we make several comparisons. Work in \[14\] offers an efficient implementation of resultant-based methods. For an alternative using Gröbner bases see \[11\]. To the best of our knowledge the only result in topology determination using isolating intervals is \[12\], where a \(O_B(N^{30})\) bound is proved.

We establish a bound of \(O_B(N^{12})\) using the isolating interval representation. It seems that the complexity in \[12\] could be improved to \(O_B(N^{10})\) using fast multiplication algorithms, fast algorithms for computations of signed sub-resultant sequences and improved bounds for the bitsize of the integers appearing in computations. To put our bounds into perspective, note that the input is in \(O_B(N^4)\), and the bitsize of all output isolation points for univariate solving is \(O_B(N^3)\), and this is tight.

The main contributions of this paper are the following: Using the aggregate separation bound, we improve the complexity for computing the sign of a polynomial evaluated over all real roots of another (lemma \[4\]). We establish a complexity bound for bivariate sign evaluation (th. \[14\]), which helps us derive bounds for root counting in an extension field (th. \[12\]) and for the problem of simultaneous inequalities (cor. \[12\]). We study the complexity of bivariate polynomial real solving, using three projection-based algorithms: a straightforward grid method (th. \[5\]), a specialized RUR approach (th. \[6\]), and an improvement of the latter using fast GCD (th. \[6\]). Our best bound is \(O_B(N^{12})\); within this bound, we also compute the root multiplicities. Computing the topology of a real plane algebraic curve is in \(O_B(N^{12})\) (th. \[5\]).

We implemented in \textsc{maple} a package for computations with real algebraic numbers and for implementing our algorithms. It is easy to use and integrates seminumerical filtering to speed up computation when the roots are well-separated. It guarantees exactness and completeness of results; moreover, the runtimes seem very encouraging. We illustrate it by experiments against well-established \textsc{C}/\textsc{C++} libraries \textsc{fgb/bs} and \textsc{synaps}. We also examine \textsc{maple} libraries \textsc{insulate} and \textsc{top}, which compute curve topology. Our software is robust and effective; its runtime is within a small constant factor w.r.t. the fastest \textsc{C}/\textsc{C++} library.

The next section presents basic results concerning real solving and operations on univariate polynomials. We extend the discussion to several variables, and focus on bivariate polynomials. The algorithms for bivariate solving and their analyses appear in sec. \[6\] followed by applications to real-root counting, simultaneous inequalities and the topology of curves. Our implementation and experiments appear in sec. \[5\]. Ancillary results and omitted proofs can be found in \[6\].

2. PRELIMINARIES

For \(f \in \mathbb{Z}[y_1,\ldots, y_i, x]\), \(dg(f)\) denotes its total degree, while \(deg_x(f)\) denotes its degree w.r.t. \(x\). \(L(f)\) bounds the bitsize of the coefficients of \(f\) (including a bit for the sign). We assume \(lg(dg(f)) = O(L(f))\). For \(a \in \mathbb{Q}\), \(L(a)\) is the maximum bitsize of numerator and denominator. Let \(M(\tau)\) denote the bit complexity of multiplying two integers of size \(\tau\), and \(M(d, \tau)\) the complexity of multiplying two univariate polynomials of degrees \(\leq d\) and coefficient bitsize \(\leq \tau\). Using FFT, \(M(\tau) = \tilde{O}(\tau)\), \(M(d, \tau) = \tilde{O}(d\tau)\).

Let \(f, g \in \mathbb{Z}[x]\), \(dg(f) = p \geq q = dg(g)\) and \(L(f), L(g) \leq \tau\). We use \(rem(f, g)\) and \(quo(f, g)\) for the Euclidean remainder and quotient, respectively. The signed polynomial remainder sequence of \(f, g\) is \(R_0 = f, R_1 = g\). \(R_{2i} = -rem(R_{2i-2}, R_{2i-1})\), \(R_{2i+1} = rem(R_{2i-1}, R_0)\). The quotient sequence contains \(Q_i = quo(R_i, R_{i+1}), i = 0\ldots k - 1\), and the quotient boot is \((Q_0,\ldots, Q_{k-1}, R_k)\).

Here, we consider signed subresultant sequences, which contain polynomials similar to the polynomials in the signed polynomial remainder sequence; see \[24\] for a unified approach to subresultants. They achieve better bounds on the coefficient bitsize and have good specialization properties. In our implementation we use Sturm-Habicht sequences, see e.g. \[13\]. By \(SR(f, g)\) we denote the signed subresultant sequence, by \(sr(f, g)\) the sequence of the principal subresultant coefficients, by \(SQ(f, g)\) the corresponding quotient boot, and by \(SR(f, g; a)\) the evaluated sequence over \(a \in \mathbb{Q}\).

If the polynomials are multivariate, then these sequences are considered w.r.t. \(x\), except if explicitly stated otherwise.

**Proposition 2.1.** \[17\] \[20\] \[20\] \[10\] \[20\] \[30\] \[30\] \[10\] \[20\] \[30\] Assuming \(p \geq q\), \(SR(f, g)\) is computed in \(O_B(p^\alpha q^{\tau})\) and \(L(SR(f, g)) = O(p\tau)\).

**Proposition 2.2.** \[17\] \[20\] \[10\] \[20\] \[30\] \[30\] \[10\] \[20\] \[30\] Let \(p \geq q\). We can compute \(SR(f, g; a)\), where \(a \in \mathbb{Q} \cup \{\pm \infty\}\) and \(L(a) = \sigma\), in \(O_B(p^{q+1}q^2p^3\sigma)\). If \(a(a)\) is known, then the bound becomes \(O_B(p^q + q^2\sigma)\).

When \(q > p\), \(SR(f, g)\) is \(f, g, -f, -(g \text{ mod } (-f))\ldots\) thus \(SR(f, g; a)\) starts with a sign variation irrespective of \(sign(g(a))\). If only the sign variations are needed, there is no need to evaluate \(g\), so prop. \[20\] yields \(O_B(p^q + q^2\sigma)\). Let \(L\) denote a list of real numbers. \(\textsc{var}(L)\) denotes the number of (possibly modified, see e.g. \[20\] [13] sign variations.

**Corollary 2.3.** For any \(f, g\), \(\textsc{var}(SR(f, g; a))\) is computed in \(O_B(p^q + q^2\sigma)\), provided \(sign(f(a))\) is known.

We choose to represent a real algebraic number \(a \in \mathbb{R}_p\) by the isolating interval representation. It includes a square-free polynomial which vanishes on \(a\) and a (rational) interval containing \(a\) and no other root.

**Proposition 2.4.** \[17\] \[10\] \[20\] \[20\] \[30\] \[30\] Let \(f \in \mathbb{Z}[x]\) have degree \(p\) and bitsize \(\tau_f\). We compute the isolating interval representation of its real roots and their multiplicities in \(O_B(p^q + p^2\tau_f)\). The endpoints of the isolating intervals have bitsize \(O(p^q + p^2\tau_f)\) and \(L(f red) = O(p^q + p\tau_f)\).

The sign of the square-free part \(f_{red}\) over the interval’s endpoints is known; moreover, \(f_{red}(a), f_{red}(b) < 0\).

**Corollary 2.5.** \[10\] \[10\] Given a real algebraic number \(a \cong (f, [a, b])\), where \(L(a) = L(b) = O(p\tau_f)\), and \(g \in \mathbb{Z}[x]\), such that \(dg(g) = q\), \(L(g) = \tau_g\), we compute \(sign(g(a))\) in bit complexity \(O_B(pq \max(\tau_f, \tau_g) + p \min(p, q)^2\tau_f)\).
Prop. 2.4 expresses the state-of-the-art in univariate root isolation. It relies on fast computation of polynomial sequences and the Davenport-Mahler bound, e.g. 55. The following lemma, derived from Davenport-Mahler’s bound, is crucial.

**Lemma 2.6 (Aggregation separation).** Given \( g \in \mathbb{Z}[x] \), the sum of the bitsize of all isolating points of the real roots of \( g \) is \( O(p^2 + p \tau) \).

We present a new complexity bound on evaluating the sign of a polynomial \( g(x) \) over a set of algebraic numbers, which have the same defining polynomial, namely over all real roots of \( f(x) \). It suffices to evaluate \( \text{SR}(f,g) \) over all the isolating endpoints of \( f \). The obvious technique, e.g. 10, is to apply cor. 2.4 \( r \) times, where \( r \) is the number of real roots of \( f \). But we can do better by applying lemma 2.6.

**Lemma 2.7.** Let \( \tau = \max\{p,\tau_1,\tau_p\} \). Assume that we have isolated the real roots of \( f \) and we know the signs of \( f \) over the isolating endpoints. Then, we can compute the sign of \( g \) over all \( r \) roots of \( f \) in \( \Omega(\mu^2 p^2\tau^2) \).

We discuss multivariate polynomials, using binary segmentation. 20. An alternative approach could be 26. Let \( f, g \in \mathbb{Z}[y_1,\ldots,y_k][x] \) with \( \deg_y(f) = p \geq q = \deg_y(g) \). \( \deg_y(f) \leq d \), and \( \deg_y(g) \leq d \). Let \( d = \prod_{i=1}^k d_i \) and \( \tau(f), \tau(g) \leq \tau \). The \( y \)-degree of every polynomial in \( \text{SR}(f,g) \), is bounded by \( \deg_y(\text{res}(f,g)) \leq (p + q)d \). Thus, the homomorphism \( \psi : \mathbb{Z}[y_1,\ldots,y_k] \to \mathbb{Z}[y] \), where

\[
y_1 \mapsto y, \quad y_2 \mapsto y^{(p+q)d_1}, \ldots, y_k \mapsto y^{(p+q)d_k-1},
\]

allows us to decode \( \text{res}(\psi(f),\psi(g)) = \text{res}(f,g) \) and obtain \( \text{res}(f,g) \). The same holds for every polynomial in \( \text{SR}(f,g) \). Now \( \psi(f), \psi(g) \in \mathbb{Z}[y][x] \) have \( y \)-degree \( \leq (p + q)^k + d \). Thus, \( \deg_y(\text{res}(\psi(f),\psi(g))) \leq (p + q)^kd \).

**Proposition 2.8.** 20 We can compute \( \text{SQ}(f,g) \), any polynomial in \( \text{SR}(f,g) \), and \( \text{res}(f,g) \) in \( \Omega(q(p+q)^{k+1}\tau d) \).

**Lemma 2.9.** \( \text{SR}(f,g) \) is computed in \( \Omega(pq(p+q)^{k+2}\tau d) \).

**Theorem 10.** We can evaluate \( \text{SR}(f,g) \) at \( x = \alpha \), where \( \alpha \in \mathbb{Q} \cup \{\infty\} \) and \( \tau = \sigma \), in \( \Omega(q(p+q)^{k+1}\tau d \max(\tau,\sigma)) \).

**Proof.** Compute \( \text{SQ}(f,g) \) in \( \Omega(pq(p+q)^{k+1}\tau d) \) then evaluate it over \( \alpha \), using binary segmentation. For this complexity we need to bound the bitsize of the resulting polynomials.

The polynomials in \( \text{SR}(f,g) \) have total degree in \( y_1,\ldots,y_k \) bounded by \( (p + q) \sum_{i=1}^k d_i \), and coefficient bitsize bounded by \( (p + q)\tau \). With respect to \( x \), the polynomials in \( \text{SR}(f,g) \) have degrees in \( \Omega(p) \), so \( \psi(x) = y \) yields values of size \( \Omega(p\tau) \). After the evaluation we obtain polynomials in \( \mathbb{Z}[y_1,\ldots,y_k] \) with coefficient bitsize bounded by \( \max(\Omega(p + q)\tau,\sigma) \leq (p + q)\max(\tau,\sigma) \).

Consider \( \chi : \mathbb{Z}[y] \to \mathbb{Z} \), such that \( y \mapsto 2^{\chi(p+q)\max(\tau,\sigma)} \) for a suitable constant \( \chi \). Apply the map \( \psi = \psi \circ \chi \) to \( f, g \). Now, \( \Omega(\psi(f)), \Omega(\psi(g)) \leq \chi(p + q)^d \max(\tau,\sigma) \). By prop. 2.8 the evaluation costs \( \Omega(pq(p+q)^{k+1}\tau d \max(\tau,\sigma)) \).

We obtain the following for \( f,g \in \mathbb{Z}[y][x] \), such that \( \deg_x(f) = p, \deg_y(g) = q, \deg_x(f), \deg_y(g) \leq d \).

**Algorithm 1:** \( \text{sign}_\text{SR}(f, \alpha, \beta) \)

```
Input: \( F \in \mathbb{Z}(x, y), \alpha \cong (A, [a_1, a_2]), \beta \cong (B, [b_1, b_2]) \)
Output: \( \text{sign}(F(\alpha, \beta)) \)
1. compute \( \text{SQ}_\text{a}(A, F) \)
2. \( L_1 \leftarrow \text{SR}_\text{a}(A, F; a_1), V_1 \leftarrow \emptyset \)
3. foreach \( f \in L_1 \) do \( V_1 \leftarrow \text{ADD}(V_1, \text{sign}_\text{SR}(f, \beta)) \)
4. \( L_2 \leftarrow \text{SR}_\text{a}(A, F; a_2), V_2 \leftarrow \emptyset \)
5. foreach \( f \in L_2 \) do \( V_2 \leftarrow \text{ADD}(V_2, \text{sign}_\text{SR}(f, \beta)) \)
6. return \( (\text{VAR}(V_1) \cup \text{VAR}(V_2)) \cdot \text{sign}(A(a)) \)
```

**Corollary 11.** We compute \( \text{SR}(f,g) \) in \( \Omega(pq(p+q)^{k+2}\tau d) \). For any polynomial, say \( \text{SR}(f,g) \), in \( \text{SR}(f,g) \), \( \deg_y(\text{SR}(f,g)) = O(\max(p,q,d)) \), \( \deg_y(\text{SR}(f,g)) = O(\max(p,q,d)) \), and also \( \text{SR}(f,g) = O(\max(p,q)^\tau) \).

**Corollary 12.** We compute \( \text{SQ}(f,g) \), any polynomial in \( \text{SR}(f,g) \), and \( \text{res}(f,g) \) in \( \Omega(pq(p+q)^{k+1}\tau d) \).

**Theorem 13.** \( \text{sign}_\text{SR}(f,g) \) is computed in \( \Omega(pq(p+q)^{k+1}\tau d) \).

**Proof.** First, we compute \( \text{SQ}_\text{a}(A, F) \) so as to evaluate \( \text{SR}(A, F) \) on the endpoints of \( \alpha \) in \( \Omega(q(p+q)^{k+1}\tau d) \).

We compute \( \text{SR}(A, F; a_1) \). The first polynomial in the sequence is \( A \), but we already know its value on \( a_1 \). This computation costs \( \Omega(pq(p+q)^{k+1}\tau d) \) by cor. 2.4 with \( q = n_1, p = n_2, d = n_1 - \tau, \sigma, \) and \( \sigma = n_2 \), where the latter corresponds to the bitsize of the endpoints. After the evaluation we obtain a list \( L_1 \), which contains \( O(n_1) \) polynomials, say \( f \in \mathbb{Z}(y) \), such that \( \deg(f) = O(n_1n_2) \). To bound the bitsize, notice that the polynomials in \( \text{SR}(f,g) \) are of degrees \( O(n_1) \) w.r.t. \( x \) and of bitsize \( O(n_2) \). After we evaluate on \( A, L_1 \), \( \text{sign}(A) = O(n_1n_2) \).

For each \( f \in L_1 \) we compute its sign over \( \beta \) and count the sign variations. We could apply directly cor. 2.4, but we can do better. If \( \deg(f) \geq n_2 \) then \( \text{SR}(B, f) = (B, f, B, g = \text{prem}(f, B), \ldots) \). We start the evaluations at \( y \) it is computed in \( \Omega(n_2) \) (prop. 2.4), \( \deg(g) = O(n_2) \) and \( \deg(g) = O(n_1n_2) \). Thus, we evaluate \( \text{SR}(B, g; a_1) \) in \( \Omega(n_2) \), by cor. 2.4 with \( p = q = n_2, \tau_1 = \sigma, \tau = n_1n_2 \). If \( \deg(f) < n_2 \) the complexity is dominated.

Since we perform \( O(n_1) \) such evaluations, all of them cost \( \Omega(\mathbb{Z}(p+q)^{k+1}\tau d) \).

We repeat for the other endpoint of \( \alpha \), subtract the sign variations, and multiply by \( \text{sign}(\alpha(a)) \), which is known from the process that isolated \( \alpha \). If the last sign in the two sequences is alternating, then \( \text{sign}(F(\alpha, \beta)) = 0 \).
3. BIVARIATE REAL SOLVING

Let \( F, G \in \mathbb{Z}[x, y] \), \( \deg(F) = \deg(G) = n \) and \( L(F) = L(G) = \sigma \). We assume relatively prime polynomials for simplicity but this hypothesis is not restrictive because it can be verified and if it does not hold, it can be imposed within the same asymptotic complexity. We study algorithms and their complexity for real solving the system \( F = G = 0 \).

The main idea is to project the roots on the \( x \) and \( y \) axes, to compute the coordinates of the real solutions and somehow to match them. The difference between the algorithms is the way they match solutions.

3.1 The grid algorithm

Algorithm Grid is straightforward, see also [14, 33]. We compute the \( x \)- and \( y \)-coordinates of the real solutions, as real roots of the resultants \( \text{res}_x(F, G) \) and \( \text{res}_y(F, G) \).

Then, we match them using the algorithm SignAt (th. 3.2) by testing all rectangles in this grid. The output is a list of pairs of real algebraic numbers represented in isolating interval representation. The algorithm also outputs rational axis-angled rectangles, guaranteed to contain a single root of the system.

To the best of our knowledge, this is the first time that the algorithm’s complexity is studied. The disadvantage of the algorithm is that exact implementation of SignAt (alg. 1) is not efficient. However, its simplicity makes it attractive. The algorithm requires no genericity assumption on the input; we study a generic shear that brings the system to generic position in order to compute the multiplicities within the same complexity bound.

The algorithm allows the use of heuristics. In particular, we may exploit easily computed bounds on the number of roots, such as the Mixed Volume or count the roots with a given abscissa \( x \) by th. 3.4.

**Theorem 3.1.** Isolating all real roots of system \( F = G = 0 \) using Grid has complexity \( \mathcal{O}_B(n^{14} + n^3 \sigma) \), provided \( \sigma = O(n^3) \).

**Proof.** First we compute the resultant of \( F \) and \( G \) w.r.t. \( y \), i.e. \( R_y \). The complexity is \( \mathcal{O}_B(n^3 \sigma) \), using cor. 2.12. Notice that \( \deg(R_y) = O(n^2) \) and \( L(R_y) = O(n \sigma) \). We isolate its real roots in \( \mathcal{O}_B(n^{12} + n^{10} \sigma) \) (prop. 2.24) and store them in \( L_y \). This complexity shall be dominated. We do the same for the \( y \) axis and store the roots in \( L_y \).

The representation of the real algebraic numbers contains the square-free part of \( R_x \) or \( R_y \). In both cases the bitsize of the polynomial is \( O(n^2 + n \sigma) \).[13, 14] The isolating intervals have endpoints of size \( O(n^2 + n \sigma) \).

Let \( r_x \), resp. \( r_y \) be the number of real roots of the corresponding resultants. Both are bounded by \( O(n^2) \). We form all possible pairs of real algebraic numbers from \( L_x \) and \( L_y \) and check for every such pair if both \( F \) and \( G \) vanish, using SignAt (th. 2.24). Each evaluation costs \( \mathcal{O}_B(n^{10} + n^3 \sigma) \) and we perform \( r_x r_y = O(n^4) \) of them. \( \square \)

We now examine the multiplicity of a root \((\alpha, \beta)\). Previous work includes [14, 30, 33]. The sum of multiplicities of all roots \((\alpha, \beta)\) equals the multiplicity of \( x = \alpha \) in the respective resultant. It is possible to apply a shear transform to the coordinate frame so as to ensure that different roots project to different points on the \( x \)-axis. We determine an adequate (horizontal) shear such that

\[
R_t(x) = \text{res}_y(F(x + ty, y), G(x + ty, y)),
\]

when \( t \rightarrow t_0 \in \mathbb{Z} \), has simple roots. \( R_{x, y} \in \mathcal{O}_B(n^{10} + n^3 \sigma) \), because a random value will typically suffice. For an alternative approach see [14], also [3]. It is straightforward to compute the multiplicities of the sheared system. Then, we need to match the latter with the roots of the original system, which is nontrivial in practice.

**Theorem 3.3.** Consider the setting of th. 3.4. Having isolated all real roots of \( F = G = 0 \), it is possible to determine their multiplicities in \( \mathcal{O}_B(n^{12} + n^{11} \sigma + n^{10} \sigma^2) \).

3.2 The \( M_{RUR} \) algorithm

The \( M_{RUR} \) algorithm assumes that the polynomials are in Generic Position: different roots project to different \( x \)-coordinates and leading coefficients w.r.t. \( y \) have no common real roots.

**Proposition 3.4.** Let \( F, G \) be co-prime polynomials, in generic position. If \( \text{SR}_j(x, y) = \text{sr}_j(x)y^j + \cdots + \text{sr}_{j-1}(x)y^{j-1} \) and \((\alpha, \beta)\) is a real solution of the system \( F = G = 0 \), then there exists \( k_j \) such that \( \text{sr}_1(\alpha) = \cdots = \text{sr}_{k_j-1}(\alpha) = 0 \), \( \text{sr}_k(\alpha) \neq 0 \) and \( \beta = \frac{\text{sr}_{k_j}(\alpha)}{\text{sr}_{k_j-1}(\alpha)} \).

This expresses the order of a solution in a Rational Univariate Representation (RUR) of the abscissa. The RUR applies to multivariate algebraic systems; it generalizes the primitive element method by Kronecker. Here we adapt it to small-dimensional systems.

Our algorithm is similar to [14, 12]. However, their algorithm computes only a RUR using prop. 2.24, so the representation of the ordinates remains implicit. Often, this representation is not sufficient (we can always compute the
We compute rationals \( \Omega \) as described in [22, 12] and define recursively polynomials \( \Gamma_j \), viz. \( \text{INTERMEDIATE POINTS}(P_j) \); the \( q_j \) have aggregate bitsize \( O(n^3 \sigma) \):
\[
q_0 < \beta_1 < q_1 < \beta_2 < \cdots < \beta_{l-1} < q_{l-1} < \beta_l < q_l,
\]
where \( \ell \leq 2n^2 \). Every \( \beta_j \) corresponds to a unique \( \alpha_j \). The multiplicity of \( \alpha_j \) as a root of \( R_x \) is the multiplicity of a real solution of the system, and this is to be ascertained.

**Sub-algorithm** \textsc{compute}_k. In order to apply prop. \[23\] for every \( \alpha_j \) we must compute \( k \in \mathbb{N}^* \) such that the intermediate points \( \alpha_j \) and \( R_x \) is the multiplicity of a real solution of the system, and this is to be ascertained.

**The \textsc{grr} algorithm**

We present an algorithm that uses some ideas from \textsc{RUR} but relies on \textsc{gcd} computations of polynomials with coefficients in an extension field to achieve efficiency (hence the name \textsc{grr}). For the \textsc{gcd} computations we use the algorithm (and the implementation) of [31].

The first step is similar to the previous algorithm: We project on the axes, we perform real solving and compute the intermediate points on the \( y \)-axis. The complexity is \( O(n^{12} + n^{10}\sigma^2) \).

For each \( z \)-coordinate, say \( \alpha_j \), we compute the square-free part \( F(\alpha, y) \) and \( G(\alpha, y) \), say \( F \) and \( G \). The complexity is that of computing the \textsc{gcd} with the derivative. In [31] this is \( O(n^{12} + n^{10}\sigma^2) \) and since we have to do it \( O(n^2) \) times, the overall cost is \( O(n^6 \sigma^2) \). Notice the bitsize of the result is \( O(n + \sigma) \).

Now for each \( \alpha_j \) we compute \( H = \text{gcd}(F, G) \). We have \( M = O(n + \sigma) \), \( N = O(n) \), \( D = O(n^2) \), \( k = O(n) \), and \( m = O(n \sigma) \). The cost is \( O(n^6 \sigma^2) \) and since we have to do it \( O(n^2) \) times, the overall cost is \( O(n^6 \sigma^2) \).

The size of \( m \) comes from Mignotte’s bound [21]. Notice that \( H \) is a square-free polynomial in \( \mathbb{Z}[\alpha]/[g] \), of degree \( O(n + \sigma) \) and \( \text{bitsize } O(n^2 + \sigma) \), the real roots of which correspond to the real solutions of the system with abscissa \( \alpha \). It should change sign only over the intervals that contain its real roots. To check these signs, we have to substitute \( y \) in \( H \) by the intermediate points, thus obtaining a polynomial in \( \mathbb{Z}[\alpha] \), of degree \( O(n) \) and \( \text{bitsize } O(n^2 + \sigma) \), \( \text{bitsize } O(n^2 + \sigma + n s_j) \), where \( s_j \) is the bitsize of the \( j \)-th intermediate point.

Now, consider this polynomial in \( \mathbb{Z}[x] \) and evaluate it over \( \alpha_j \). Using cor. [24] with \( p = n^2 \), \( r_j = n^2 + n \sigma \), \( q = n \), and \( r_j = n^2 + n \sigma + n s_j \), this costs \( O(n^6 + n^5\sigma + n^4 s_j) \). Summing over \( O(n^2) \) points and using lem. [24] we obtain \( O(n^6 + n^5\sigma) \). Thus, the overall complexity is \( O(n^6 + n^5\sigma) \).

**Theorem 3.5.** We can isolate all real roots of \( F = G = 0 \), if \( F, G \) are in generic position, by \textsc{mrr} in \( O(n^{12} + n^{10}\sigma) \).

We conclude that \( \text{SIGN}(B_j, \alpha) \) and \( \text{FIND} \) have complexity \( O(n^{10} + n^7\sigma) \). As for the overall complexity of the loop (Lines 6--11) the complexity is \( O(n^{10} + n^7\sigma) \), since it is executed \( O(n^2) \) times.

**Theorem 3.6.** We can isolate the real roots of the system \( F = G = 0 \), using \textsc{grr} in \( O(n^{12} + n^{10}\sigma) \).
4. APPLICATIONS

Real root counting. We wish to count the number of roots of \( F = F(a,y) \in (\mathbb{Z}[a])[y] \) in \( \mathbb{R} \), in \((c, +\infty)\) and in \((\beta, +\infty)\). Assume \( \alpha, \beta \in \mathbb{R}_{\text{alg}} \) as above, but with \( \mathcal{L}(A), \mathcal{L}(B) \leq \tau \) and \( \epsilon \in \mathbb{Q} \), such that \( \mathcal{L}(c) = \lambda \). Moreover, let \( n_1^2 = \mathcal{O}(n_2) \), as will be the case in applications.

THEOREM 4.1. We count the real roots of \( F \) in \((c, +\infty)\), \((\beta, +\infty)\) and \((c, +\infty)\), respectively, in \( \mathcal{O}(n_1^n + n_1^n + n_2) \), \( \mathcal{O}(\lambda n_2^2 + \lambda n_2^2 + \gamma) \) and \( \mathcal{O}(\epsilon n_2^2 + \epsilon n_2^2 + \delta, \epsilon, \lambda, \gamma) \).

The proof uses Sturm’s theorem and the good specialization properties of subresultants in order to switch the order of substitution \( x = \alpha \) and sequence computation; see [22].

Simultaneous inequalities in two variables. Let \( P, Q, A_1, \ldots, A_{\ell_1}, B_1, \ldots, B_{\ell_2}, C_1, \ldots, C_{\ell_3} \in \mathbb{Z}[X,Y] \), such that their total degrees are bounded by \( n \) and their bitsize by \( \sigma \). We wish to compute \((\alpha, \beta) \in \mathbb{R}_{\text{alg}}^2 \) such that \( P(\alpha, \beta) = Q(\alpha, \beta) = 0 \) and also \( A_i(\alpha, \beta) > 0 \) and \( B_i(\alpha, \beta) < 0 \) and \( C_\ell(\alpha, \beta) = 0 \), where \( 1 \leq i \leq \ell_1, 1 \leq j \leq \ell_2, 1 \leq k \leq \ell_3 \). Let \( \ell = \ell_1 + \ell_2 + \ell_3 \).

COROLLARY 4.2. There is an algorithm that solves the problem of \( \epsilon \) simultaneous inequalities of degree \( \leq n \) and bitsize \( \leq \sigma \), in \( \mathcal{O}(\epsilon n_1^p + \epsilon n_1^p + n^3 + n^3 + n^3) \).

The complexity of topology. We improve the complexity of computing the topology of a real plane algebraic curve. See [22] for the algorithm.

We consider the curve, in generic position, defined by \( F \in \mathbb{Z}[x,y] \), such that \( \deg(F) = n \) and \( \mathcal{L}(F) = \sigma \). We compute the critical points of the curve, i.e. solve \( F = F_y = 0 \) in \( \mathcal{O}(n_2^2 + n_3^2) \). Next, we compute the intermediate points on the x-axis, in \( \mathcal{O}(n^3 + n^3) \) (lem. 22). For each intermediate point, say \( q_j \), we need to compute the number of branches of the curve that cross the vertical line \( x = q_j \). This is equivalent to computing the number of real solutions of the polynomial \( F(q_j, y) \in \mathbb{Z}[y] \), which has degree \( d \) and bitsize \( \mathcal{O}(\sigma \lg(q_j)) \). For this we use Sturm’s theorem and th. 2.2 (and the cost is \( \mathcal{O}(n^3 \mathcal{L}(q_j)) \)). For all \( q_j \)’s the cost is \( \mathcal{O}(n^2 + n^3 + n^3) \).

For each critical point, say \((\alpha, \beta)\), we need to compute the number of branches of the curve that cross the vertical line \( x = \alpha \) and the number of them that are above \( y = \beta \). The first task corresponds to computing the number of real roots of \( F(\alpha, y) \), by application of th. 24 in \( \mathcal{O}(n_2^3 + n^3) \), where \( n_1 = n, n_2 = n^2, \tau = n^3 + n \). Since there are \( \mathcal{O}(n^2) \) critical values, the overall cost of the step is \( \mathcal{O}(n_1^3 + n_1^3 + n_1^3) \).

Finally, we compute the number of branches that cross the line \( x = \alpha \) and are above \( y = \beta \). We do this by th. 24 in \( \mathcal{O}(n_2^3 + n_1^3) \). Since there are \( \mathcal{O}(n^2) \) critical points, the complexity is \( \mathcal{O}(n_1^3 + n_1^3 + n_1^3) \). It remains to connect the critical points according to the information that we have for the branches. The complexity of this step is dominated. It now follows that the complexity of the algorithm is \( \mathcal{O}(n_1^3 + n_1^3 + n_1^3) \), or \( \mathcal{O}(n_1^3) \), which is worse by a factor than \( \mathcal{O}(n^3) \).

We improve the complexity of the last step since \( \text{M} \text{RUR} \) computes the RUR representation of the ordinates. Thus, instead of performing bivariate sign evaluations in order to compute the number of branches above \( y = \beta \), we can substitute the RUR representation of \( \beta \) and perform univariate sign evaluations. This corresponds to computing the sign of \( \mathcal{O}(n^2) \) polynomials of degree \( \mathcal{O}(n^3) \) and bitsize \( \mathcal{O}(n^4 + n^3) \), over all the \( a \)'s [22]. Using lem. 27 for each polynomial the cost is \( \mathcal{O}(n_1^3 + n_1^3) \), and since there are \( \mathcal{O}(n_1^3) \) of them, the total cost is \( \mathcal{O}(n_1^3 + n_1^3) \).

THEOREM 4.3. We compute the topology of a real plane algebraic curve, defined by a polynomial of degree \( n \) and bitsize \( \sigma \), in \( \mathcal{O}(n_1^3 + n_1^3 + n_1^3) \).

Thus the overall complexity of the algorithm improves the previously known bound by a factor of \( \mathcal{O}(n^2) \). We assumed generic position, since we can apply a shear to achieve this, see sec. 4.4.

5. IMPLEMENTATION AND EXPERIMENTS

We describe our open source \textsc{maple} implementation\footnote{www.di.uoa.gr/~erga/soft/SLV\_index.html} and illustrate its capabilities through comparative experiments. The design is based on object oriented programming and the generic programming paradigm in view of transferring our implementation to \textsc{C++}.

The class of real algebraic numbers represents them in isolating interval representation. We provide various algorithms for computing signed polynomial remainder sequences; real solving univariate polynomials using Sturm’s algorithm; computations with one and two real algebraic numbers, such as sign evaluation, comparison; and our algorithms for real solving of bivariate systems. Computations are performed first using intervals with floating point arithmetic and, if they fail, then an exact algorithm using rational arithmetic is called. For \textsc{gcd} computations in an extension field we use the \textsc{maple} package of [11]. We have not implemented, yet, the optimal algorithms for computing and evaluating polynomial remainder sequences.

Overall performance results are shown on tab. 11 averaged over 10 iterations. Systems \( R_i, M_i, D_i \) are presented in [11], systems \( C_i \) in [13], and \( W_i \) are the \( C_i \) after swapping the \( x \) and \( y \) variables. For the first data set, there are no timings for \textsc{insulate} and \textsc{top} since it was not easy to modify their code so as to deal with general polynomial systems. The rest correspond to algebraic curves, i.e. polynomial systems of the form \( f = f_y = 0 \), that all packages can deal with.

It seems that \textsc{gur} is our solver of choice since it is faster than \textsc{grid} and \textsc{murr} in 17 out of our 18 instances. However, this may not hold when the extension field is of high degree. \textsc{gur} yields solutions in less than a second, apart from system \( C_5 \). Overall, for total degrees \( \leq 8 \), \textsc{gur} requires less than 0.4 secs to send. As a result, \textsc{gur} is 7-11 times faster than \textsc{grid}, and about 38 times than \textsc{murr}. One reason is that the sheared systems that \textsc{murr} solves are dense and of increased bitsize.

Among our algorithms, \textsc{grid} and \textsc{murr} benefit the most from filtering. \textsc{gur} gains only a factor of 1.1-2. \textsc{grid} gains a factor of 2-5. In \textsc{murr} we use one more filtering heuristic technique: after computing the intermediate points on the \( y \)-axis, we perform refining with \textsc{grid} (up to 20 times on systems with high degree) on the intervals of the candidate solutions along the \( x \)-axis. Recall that \textsc{murr} binary-searches for solutions along the \( y \)-axis. The refinement must not be excessive since this will increase the bitsize of the coefficients. This has been very efficient in practice, resulting on average an additional speedup of 2.2-3.4; overall filtering improves \textsc{murr} by a factor of 7-11.
If a polynomial system did not comply with the generic position criterion required by \texttt{M\_RUR}, we deterministically tested a value for the required shear; in all cases our first candidate ($t = 3$) worked. This is relatively inexpensive on systems with polynomials of degree $\leq 5$. For systems with polynomials of higher degree, in some cases the deterministic shear computation is more expensive than real solving. Hence, a random shear is more efficient in general, as suggested also by the asymptotic analysis.

We tested \texttt{FGb/Rs} \cite{23}, which performs exact real solving using Gröbner bases and RUR, through its \textsc{maple} interface. It should be underlined that communication with \textsc{maple} increases the runtimes. \texttt{G\_RUR} is faster in 8 out of the 18 instances, including the difficult system $C_3$. Lastly, we examined 3 \textsc{synaps} solvers: \textsc{sturm} is a naive implementation of \textsc{grgb} \cite{21}; \textsc{subdiv} implements \textsc{grgb}, and is based on Bernstein basis and \textsc{double} arithmetic. It needs an initial box for computing the real solutions of the system and in all the cases we used $[-10,10] \times [-10,10]$. \textsc{newmac} is a general purpose solver based on computations of generalized eigenvectors using \textsc{lapack}, which computes all complex solutions. \textsc{sturm} is faster than our \textsc{maple} implementation of \textsc{grgb}. \textsc{subdiv} is faster than all of our solvers in 6, and \textsc{newmac} in 16, of the 18 systems.

We also tested other \textsc{maple} implementations: \textsc{insulate} is a package that implements \textsc{grgb} for computing the topology of real algebraic curves, and \textsc{top} implements \textsc{grgb}. Both packages were kindly provided to us by their authors. We tried to modify the packages so as to stop them as soon as they compute the real solutions of the corresponding bivariate system. \textsc{top} has an additional parameter that sets the initial precision (digits). A very low initial precision or a very high one results in inaccuracy or performance loss; but there is no easy way for choosing a good value. Hence, we followed \cite{11} and recorded its performance on initial values of 60 and 500 digits. Compared to \texttt{G\_RUR}, \textsc{insulate} is 2-46 times slower when the total degree is $\geq 6$. On the other hand, \textsc{top} is slower than \texttt{G\_RUR} 1.7-23 times when the total degree is $\geq 6$ and the curves have many critical points.

We underline that we do not consider experiments as competition, but a crucial step for improving existing software. Moreover, it is very difficult to compare different packages, since in most cases they are made for different needs. In addition, accurate timing in \textsc{maple} is delicate.

\textsc{insulate} has demonstrated more robust behaviour than \textsc{top}, especially when the latter is used with low precision. \textsc{grid} could not find a solution within 20 minutes even when we increased the default \textsc{maple} stack size. In \texttt{FGb/Rs}'s case, some errors regarding the communication of the application with the \textsc{maple} kernel occurred. \textsc{sturm} failed to reply within our time limits for $C_5$. As for \textsc{newmac} and \textsc{subdiv}, some numerical errors are introduced since the former is based on \textsc{lapack} and the latter on floating point arithmetic. \textsc{subdiv} and \textsc{newmac} fail to compute the correct number of real solutions in at least half of the cases. Finally, \textsc{sturm}'s inefficiency, in some experiments, is basically due to the lack of modular algorithms for computing resultants.

\textsc{grid} and \texttt{M\_RUR} demonstrate a high fluctuation in runtimes, compared, e.g. to the stability of \textsc{newmac} or \texttt{FGb/Rs}. The latter spends a lot of time on Gröbner bases. The rest of the solvers demonstrate a similar fluctuation, especially those that are based on \textsc{maple}.

To summarize, we believe that the implementation of our algorithms gives very encouraging results, at least for polynomial systems of moderate degree.

The time that each algorithm spends on the various steps is on tab. \ref{table1} as percentages of the overall computing times in tab. \ref{table1}. \textsc{Projections} shows the time for the computa-

\begin{table}[h]
\centering
\begin{tabular}{||c|c|c|c|c|c|c|c|c|c|c|c|c|c|c||}
\hline
\hline
system & deg & \multicolumn{11}{c||}{Average Time (msecs)} \\
& & \multicolumn{3}{c|}{BIVARIATE SOLVING} & & \multicolumn{3}{c|}{TOPOLOGY} \\
& & this paper (SLV) & \textsc{FGb/Rs} & \textsc{sturm} & \textsc{subdiv} & \textsc{newmac} & Insulate & Top \\
& & \textsc{grid} & \textsc{m\_rur} & \textsc{g\_rur} & & & & & & & \textsc{G\_RUR} & \textsc{maple} & \textsc{FGb/Rs} & \textsc{sturm} & \textsc{subdiv} & \textsc{newmac} & Insulate & Top \\
\hline
$R_2$ & 3 & 1 & 1 & 60 & 21 & 36 & 24 & 1 & 1 & 1 & - & - & - & - & - & - & - & - \\
$M_1$ & 3 & 3 & 4 & 8 & 7 & 2 & 22 & 1 & 1 & 1 & - & - & - & - & - & - & - & - \\
$C_1$ & 7 & 6 & 6 & 1 & 8 & 6 & 954 & 222 & 93 & 479 & 170 & 265 & 39 & 524 & 409 & 1,367 & - & - & - \\
\hline
\end{tabular}
\caption{Performance averages over 10 runs in \textsc{maple} 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.}
\label{table1}
\end{table}
tion of the resultants, **Univ. Solving** for real solving the resultants, and **Sorting** for sorting solutions. In **GRID**’s and **MRUR**’s case, **Biv. Solving** corresponds to matching. In **GRID**’s case timings for matching are divided between **Rational Biv.** and **Ralg Biv.**: the first refers to when at least one of the co-ordinates is a rational number, while the latter indicates timings when both co-ordinates are not rational. **Inter. Points** refers to computation of the intermediate points between resultant roots along the y-axis. **StHa Seq.** refers to the computation of the **StHa** sequence. **Filter x-cand** shows the time for additional filtering. **Compute K** reflects the time for sub-algorithm **Compute-K**.

In a nutshell, **GRID** spends more than 73% of its time in matching. Recall that this percent includes the application of filters. **MRUR** spends about 45-50% of its time in matching and about 24-27% in the pre-computation filtering technique. **GRID** runs 55-80% of its time in matching, including gcd computations in an extension field.

**Acknowledgements.** We acknowledge discussions with B. Mourrain, thank J-P. Pavone for his help with SYNAPS, and the anonymous referees for their comments. The third author thanks F. Rouillier for various comments.

### 6. REFERENCES


