On the asymptotic and practical complexity of solving bivariate systems over the reals

Dimitrios I. Diochnos

University of Illinois at Chicago, USA

Ioannis Z. Emiris

National Kapodistrian University of Athens, HELLAS

Elias P. Tsigaridas

INRIA Sophia-Antipolis, FRANCE

Abstract

This paper is concerned with exact real solving of well-constrained, bivariate polynomial systems. The main problem is to isolate all common real roots in rational rectangles, and to determine their intersection multiplicities. We present three algorithms and analyze their asymptotic bit complexity, obtaining a bound of $\widetilde{\mathcal{O}}_B(N^{14})$ for the purely projection-based method, and $\widetilde{\mathcal{O}}_B(N^{12})$ for two subresultant-based methods: this notation ignores polylogarithmic factors, where N bounds the degree, and the bitsize of the polynomials. The previous record bound was $\widetilde{\mathcal{O}}_B(N^{14})$.

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

All algorithms have been implemented in MAPLE, in conjunction with numeric filtering. We compare them against FGB/RS, system solvers from SYNAPS, and MAPLE libraries INSULATE and TOP, which compute curve topology. Our software is among the most robust, and its runtimes are comparable, or within a small constant factor, with respect to the C/C++ libraries.

Key words: real solving, polynomial system, complexity, MAPLE software

Email addresses: diochnos (AT) math.uic.edu (Dimitrios I. Diochnos), emiris (AT) di.uoa.gr (Ioannis Z. Emiris), elias.tsigaridas (AT) sophia.inria.fr (Elias P. Tsigaridas).

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 suppose relatively prime polynomials for simplicity, but this hypothesis is not restrictive. A question of independent interest, which we tackle, is to compute the topology of a real plane algebraic curve.

Our algorithms isolate all common real roots inside non-overlapping rational rectangles, output them as pairs of algebraic numbers, and determine the intersection multiplicity per root. Algebraic numbers are represented by an isolating interval and a square-free polynomial.

In this paper, $\mathcal{O}_B$ means bit complexity and $\widetilde{\mathcal{O}}_B$ means that we are ignoring polylogarithmic factors. We derive a bound of $\widetilde{\mathcal{O}}_B(N^{12})$, whereas the previous record bound was $\widetilde{\mathcal{O}}_B(N^{14})$ [16], see also [3], derived from the closely related problem of computing the topology of 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, and is used in applications. In [16], it is stated that "isolating intervals provide worst [sic] bounds". 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 binary segmentation. We exploit the recent advances on univariate root isolation, which reduced complexity by one to three orders of magnitude, to $\widetilde{\mathcal{O}}_B(N^6)$ [11, 12, 15]. This brought complexity closer to $\widetilde{\mathcal{O}}_B(N^4)$, which is achieved by numerical methods [26].

In [19], 2×2 systems are solved and the multiplicities computed under the assumption that a generic shear has been obtained, based on [32]. In [36], 2×2 systems of bounded degree were studied, obtained as projections of the arrangement of 3D quadrics. This algorithm is a precursor of ours, see also [14], except that matching and multiplicity computation was simpler. In [23], 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., [22, 27].

For determining the topology of a real algebraic plane curve, the best bound is $\widetilde{\mathcal{O}}_B(N^{14})$ [3, 16]. In [37] three projections are used; this is implemented in INSULATE, with which we make several comparisons. Work in [13] offers an efficient implementation of resultant-based methods, whereas Gröbner bases are employed in [7]. To the best of our knowledge, the only result for topology determination using isolating intervals is [2], where a $\widetilde{\mathcal{O}}_B(N^{30})$ bound is proved.

We establish a bound of $\widetilde{\mathcal{O}}_B(N^{12})$ using the isolating interval representation. It seems that the complexity in [16] could be improved to $\widetilde{\mathcal{O}}_B(N^{10})$ using fast multiplication algorithms, fast algorithms for computations of signed subresultant sequences, and improved bounds for the bitsize of the integers appearing in computations. To put our bounds

into perspective, the input size is in $\mathcal{O}_B(N^3)$, and the total bitsize of all output isolation points for univariate solving is in $\widetilde{\mathcal{O}}_B(N^2)$, and this is tight. Notice that lower bounds in real algebraic geometry refer almost exclusively to arithmetic complexity [5].

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 (lem. 7). We establish a complexity bound for bivariate sign evaluation (th. 14), which helps us derive bounds for root counting in an extension field (lem. 21) and for the problem of simultaneous inequalities (cor. 24). We study the complexity of bivariate polynomial real solving, using three projection-based algorithms: a straightforward grid method (th. 15), a specialized RUR (Rational Univariate Representation) approach (th. 19), and an improvement of the latter using fast GCD (th. 20). Our best bound is $\widetilde{\mathcal{O}}_B(N^{12})$; within this bound, we also compute the root multiplicities. Computing the topology of a real plane algebraic curve is in $\widetilde{\mathcal{O}}_B(N^{12})$ (th. 25).

We implemented in 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 are quite encouraging. We illustrate it by experiments against well-established `C`/`C++` libraries FGB/RS and SYNAPS. We also examine MAPLE libraries INSULATE and TOP, which compute curve topology. Our software is among the most robust; its runtime is within a small constant factor with respect to the fastest `C`/`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. 4, followed by applications to real-root counting, simultaneous inequalities and the topology of curves. Our implementation and experiments appear in sec. 6.

A preliminary version of our results appeared in [9].

2. Univariate polynomials

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

Let $f, g \in \mathbb{Z}[x]$, $\mathsf{dg}(f) = p \geq q = \mathsf{dg}(g)$ and $\mathcal{L}(f), \mathcal{L}(g) \leq \tau$. We use $\mathsf{rem}(f, g)$ and $\mathsf{quo}(f, g)$ for the Euclidean remainder and quotient, respectively. The *signed polynomial remainder sequence* of f, g is $R_0 = f$, $R_1 = g$, $R_2 = -\mathsf{rem}(f, g)$, $\ldots$, $R_k = -\mathsf{rem}(R_{k-2}, R_{k-1})$, where $\mathsf{rem}(R_{k-1}, R_k) = 0$. The *quotient sequence* contains $Q_i = \mathsf{quo}(R_i, R_{i+1})$, $i = 0 \ldots k-1$, and the *quotient boot* is $(Q_0, \ldots, Q_{k-1}, R_k)$.

We consider signed subresultant sequences [3], which contain polynomials similar to the polynomials in the signed polynomial remainder sequence; see [35] 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 (or Sylvester-Habicht) sequences, see e.g. [3, 18, 20]. By $\mathbf{SR}(f, g)$ we denote the signed subresultant sequence, by $\mathbf{sr}(f, g)$ the sequence of the principal subresultant coefficients, by

$\mathbf{SRQ}(f,g)$ the corresponding quotient boot. By $\mathbf{SR}_j(f,g)$, or simply $\mathbf{SR}_j$ if the corresponding polynomials can be easily deduced from the context, we denote an element of the sequence; similarly for $\mathbf{sr}_j$ and $\mathbf{SRQ}_j$. Finally, by $\mathbf{SR}(f,g;\mathsf{a})$ we denote the evaluated sequence over $\mathsf{a}\in\mathbb{Q}$. If the polynomials are multivariate, then these sequences are considered w.r.t. x, except if explicitly stated otherwise.

Proposition 1. [20, 28] Assuming $p\geq q$, $\mathbf{SR}(f,g)$ is computed in $\widetilde{\mathcal{O}}_B(p^2q\tau)$ and $\mathcal{L}\left(\mathbf{SR}_j(f,g)\right)=\mathcal{O}(p\tau)$. For any f,g, their quotient boot, any polynomial in $\mathbf{SR}(f,g)$, their resultant, and their gcd are computed in $\widetilde{\mathcal{O}}_B(pq\tau)$.

The following proposition is a slightly modified version of the one that appeared in [20, 28].

Proposition 2. Let $p\geq q$. We can compute $\mathbf{SR}(f,g;\mathsf{a})$, where $\mathsf{a}\in\mathbb{Q}\cup\{\pm\infty\}$ and $\mathcal{L}\left(\mathsf{a}\right)=\sigma$, in $\widetilde{\mathcal{O}}_B(pq\tau+q^2\sigma+p^2\sigma)$. If $f(\mathsf{a})$ is known, then the bound becomes $\widetilde{\mathcal{O}}_B(pq\tau+q^2\sigma)$.

Proof. Let $\mathbf{SR}_{q+1}=f$ and $\mathbf{SR}_q=g$. For the moment we forget $\mathbf{SR}_{q+1}$. We may assume that $\mathbf{SR}_{q-1}$ is computed, since the cost of computing one element of $\mathbf{SR}(f,g)$ is the same as that of computing $\mathbf{SRQ}(f,g)$ (Pr. 1) and we consider the cost of evaluating the sequence $\mathbf{SR}(g,\mathbf{SR}_{q-1})$ on a.

We follow Lickteig and Roy [20]. For two polynomials A,B of degree bounded by D and bitsize bounded by L, we can compute $\mathbf{SR}(A,B;\mathsf{a})$, where $\mathcal{L}\left(\mathsf{a}\right)\leq L$, in $\widetilde{\mathcal{O}}_B(\mathsf{M}\left(D,L\right))$. In our case $D=\mathcal{O}(q)$ and $L=\mathcal{O}(p\tau+q\sigma)$, thus the total cost is $\widetilde{\mathcal{O}}_B(pq\tau+q^2\sigma)$.

It remains to compute the evaluation $\mathbf{SR}_{q+1}(\mathsf{a})=f(\mathsf{a})$. This can be done using Horners' scheme in $\widetilde{\mathcal{O}}_B(p\max\{\tau,p\sigma\})$. Thus, the whole procedure has complexity

$$\widetilde{\mathcal{O}}_B(pq\tau+q^2\sigma+p\max\{\tau,p\sigma\}),$$

where the term $p\tau$ is dominated by $pq\tau$. □

When $q>p$, $\mathbf{SR}(f,g)$ is $f,g,-f,-(g \bmod (-f))\ldots$, thus $\mathbf{SR}(f,g;\mathsf{a})$ starts with a sign variation irrespective of $\mathrm{sign}(g(\mathsf{a}))$. If only the sign variations are needed, there is no need to evaluate g, so prop. 2 yields $\widetilde{\mathcal{O}}_B(pq\tau+p^2\sigma)$. Let L denote a list of real numbers. $\mathtt{VAR}(L)$ denotes the number of (possibly modified, see e.g. [3, 16, 18]) sign variations.

Corollary 3. For any f,g, $\mathtt{VAR}(\mathbf{SR}(f,g;\mathsf{a}))$ is computed in $\widetilde{\mathcal{O}}_B(pq\tau+\min\{p,q\}^2\sigma)$, provided $\mathrm{sign}(f(\mathsf{a}))$ is known.

We choose to represent a real algebraic number $\alpha\in\mathbb{R}_{alg}$ by the *isolating interval* representation. It includes a square-free polynomial which vanishes on α and a (rational) interval containing α and no other root. By f_{red} we denote the square-free part of f.

Proposition 4. [11, 12, 15] Let $f\in\mathbb{Z}[x]$ have degree p and bitsize τ_f. We compute the isolating interval representation of its real roots and their multiplicities in $\widetilde{\mathcal{O}}_B(p^6+p^4\tau_f^2)$. The endpoints of the isolating intervals have bitsize $\mathcal{O}(p^2+p\,\tau_f)$ and $\mathcal{L}\left(f_{red}\right)=\mathcal{O}(p+\tau_f)$.

Notice that after real root isolation, the sign of the square-free part f_{red} over the interval's endpoints, say $[\mathsf{a},\mathsf{b}]$ is known; moreover, $f_{red}(\mathsf{a})f_{red}(\mathsf{b})<0$. The following proposition takes advantage of this fact and is a refined version of similar proposition in e.g. [3, 15].

Corollary 5. Given a real algebraic number $\alpha \cong (f, [\mathsf{a}, \mathsf{b}])$, where $\mathcal{L}(\mathsf{a}) = \mathcal{L}(\mathsf{b}) = \mathcal{O}(p\tau_f)$, and $g \in \mathbb{Z}[x]$, such that $\mathsf{dg}(g) = q$ and $\mathcal{L}(g) = \tau_g$, we can compute $\text{sign}(g(\alpha))$ in $\widetilde{\mathcal{O}}_B(pq \max\{\tau_f, \tau_g\} + p \min\{p, q\}^2 \tau_f)$.

Proof. Assume that α is not a common root of f and g in $[\mathsf{a}, \mathsf{b}]$, then it is known that

$$\text{sign}(g(\alpha)) = [\mathtt{VAR}(\mathbf{SR}(f, g; \mathsf{a})) - \mathtt{VAR}(\mathbf{SR}(f, g; \mathsf{b}))] \, \text{sign}(f'(\alpha)).$$

Actually the previous relation holds in a more general context, when f *dominates* g, see [38] for details. Notice that $\text{sign}(f'(\alpha)) = \text{sign}(f(\mathsf{b})) - \text{sign}(f(\mathsf{b}))$, which is known from the real root isolation process.

The complexity of the operation is dominated by the computation of $\mathtt{VAR}(\mathbf{SR}(f, g; \mathsf{a}))$ and $\mathtt{VAR}(\mathbf{SR}(f, g; \mathsf{b}))$, i.e. we compute **SRQ** and evaluate it on a and b.

As explained above, there is no need to evaluate the polynomial of the largest degree, i.e. the first (and the second if $p < q$) of $\mathbf{SR}(f, g)$ over a and b. The complexity is that of cor. 3, i.e. $\widetilde{\mathcal{O}}_B(pq \max\{\tau_f, \tau_g\} + \min\{p, q\}^2 p \, \tau_f)$. Thus the operation costs two times the complexity of the evaluation of the sequence over the endpoints of the isolating interval.

If α is a common root of f and g, or if f and g are not relative prime, then their gcd, which is the last non-zero polynomial in $\mathbf{SR}(f, g)$ is not a constant. Hence, we evaluate **SR** on a and b, we check if the last polynomial is not a constant and if it changes sign on a and b. If this is the case, then $\text{sign}(g(\alpha)) = 0$. Otherwise we proceed as above. □

Prop. 4 expresses the state-of-the-art in univariate root isolation. It relies on fast computation of polynomial sequences and the Davenport-Mahler-Mignotte bound, see [8] for the first version of this bound. The following lemma, a direct consequence of Davenport-Mahler-Mignotte bound, is crucial.

Lemma 6 (Aggregate separation)**.** Given $f \in \mathbb{Z}[x]$, the sum of the bitsize of *all* isolating points of the real roots of f is $\mathcal{O}(p^2 + p \, \tau_f)$.

Proof. Let there be $r \leq p$ real roots. The isolating point, computed by a real root isolation subdivision algorithm [11, 12, 15], between two consecutive real roots, say α_j and α_{j+1}, is of magnitude at most $\frac{1}{2}|\alpha_j - \alpha_{j+1}| := \frac{1}{2}\Delta_j$. Thus their product is $\frac{1}{2^r} \prod_{j=1}^{r-1} \Delta_j$. Using the Davenport-Mahler-Mignotte bound, the product is bounded from below, that is $\prod_j \Delta_j \geq 2^{-\mathcal{O}(p^2 + p\tau_f)}$. Taking logarithms, we conclude the proof. □

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 $\mathbf{SR}(f, g)$ over all the isolating endpoints of f. The obvious technique, e.g. [15], see also [3, 31], is to apply cor. 5 r times, where r is the number of real roots of f. But we can do better by applying lem. 6:

Lemma 7. Let $\tau = \max\{p, \tau_f, \tau_g\}$. Assume that we have isolated the r 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 $\widetilde{\mathcal{O}}_B(p^2 q \tau)$.

Proof. Let s_j be the bitsize of the j-th endpoint, where $0 \leq j \leq r$. The evaluation of $\mathbf{SR}(f, g)$ over this endpoint, by cor. 3, costs $\widetilde{\mathcal{O}}_B(pq\tau + \min\{p, q\}^2 s_j)$. To bound the overall cost, we sum over all isolating points. The first summand is $\widetilde{\mathcal{O}}_B(p^2 q \tau)$. By prop. 6, the second summand becomes $\widetilde{\mathcal{O}}_B(\min\{p, q\}^2 (p^2 + p\tau_f))$ and is dominated. □

3. Multivariate polynomials

In this section, we extend the results of the previous section to multivariate polynomials, using binary segmentation [28]. Let $f, g \in (\mathbb{Z}[y_1, \ldots, y_k])[x]$ with $\mathtt{dg}_x(f) = p \geq q = \mathtt{dg}_x(g)$, $\mathtt{dg}_{y_i}(f) \leq d_i$ and $\mathtt{dg}_{y_i}(g) \leq d_i$. Let $d = \prod_{i=1}^{k} d_i$ and $\mathcal{L}(f), \mathcal{L}(g) \leq \tau$. The y_i-degree of every polynomial in $\mathbf{SR}(f, g)$ is bounded by $\mathtt{dg}_{y_i}(\mathtt{res}(f, g)) \leq (p+q)d_i$. Thus, the homomorphism $\psi : \mathbb{Z}[y_1, \ldots, y_k] \to \mathbb{Z}[y]$, where

$$y_1 \mapsto y,\ y_2 \mapsto y^{(p+q)d_1},\ \ldots,\ y_k \mapsto y^{(p+q)^{k-1}d_1 \cdots d_{k-1}},$$

allows us to decode $\mathtt{res}(\psi(f), \psi(g)) = \psi(\mathtt{res}(f, g))$ and obtain $\mathtt{res}(f, g)$. The same holds for every polynomial in $\mathbf{SR}(f, g)$. Notice that $\psi(f), \psi(g) \in (\mathbb{Z}[y])[x]$ have $y-$degree less or equal to $(p+q)^{k-1}d$ since, in the worst case, f or g contains a monomial of the form $y_1^{d_1} y_2^{d_2} \ldots y_k^{d_k}$. Thus, $\mathtt{dg}_y(\mathtt{res}(\psi(f), \psi(g))) < (p+q)^k d$.

Proposition 8. [28] We can compute $\mathbf{SRQ}(f, g)$, any polynomial in $\mathbf{SR}(f, g)$, and $\mathtt{res}(f, g)$ w.r.t. x in $\widetilde{\mathcal{O}}_B(q(p+q)^{k+1}d\tau)$.

Lemma 9. We can compute $\mathbf{SR}(f, g)$ in $\widetilde{\mathcal{O}}_B(q(p+q)^{k+2}d\tau)$.

Proof. Every polynomial in $\mathbf{SR}(f, g)$ has coefficients of magnitude bounded $2^{c(p+q)\tau}$, for a suitable constant c, assuming $\tau > \lg(d)$. Consider the map $\chi : \mathbb{Z}[y] \mapsto \mathbb{Z}$, where $y \mapsto 2^{\lceil c(p+q)\tau \rceil}$, and let $\phi = \psi \circ \chi : \mathbb{Z}[y_1, y_2 \ldots, y_k] \to \mathbb{Z}$. Then $\mathcal{L}(\phi(f)), \mathcal{L}(\phi(g)) \leq c(p+q)^k d\tau$. Now apply prop. 1.

In order to complete the computation we should recover the result from the computed sequence, that is to apply the inverse image of ϕ. The cost of this computation (almost linear w.r.t. the output) is dominated; which is always the case. □

Theorem 10. We can evaluate $\mathbf{SR}(f, g)$ at $x = \mathtt{a}$ where $\mathtt{a} \in \mathbb{Q} \cup \{\infty\}$ and $\mathcal{L}(\mathtt{a}) = \sigma$, in $\widetilde{\mathcal{O}}_B(q(p+q)^{k+1}d\max\{\tau, \sigma\})$.

Proof. First we compute $\mathbf{SRQ}(f, g)$ in $\widetilde{\mathcal{O}}_B(q(p+q)^{k+1}d\tau)$ (prop. 8), and then we evaluate the sequence over $\mathtt{a}$, using binary segmentation. For the latter we need to bound the bitsize of the resulting polynomials.

The polynomials in $\mathbf{SR}(f, g)$ 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 $\mathbf{SR}(f, g)$ have degrees in $\mathcal{O}(p)$, so substitution $x = \mathtt{a}$ yields values of size $\widetilde{\mathcal{O}}(p\sigma)$. After the evaluation we obtain polynomials in $\mathbb{Z}[y_1, \ldots, y_k]$ with bitsize bounded by $\max\{(p+q)\tau, p\sigma\} \leq (p+q)\max\{\tau, \sigma\}$.

Consider the map $\chi : \mathbb{Z}[y] \to \mathbb{Z}$, where $y \mapsto 2^{\lceil c(p+q)\max\{\tau,\sigma\} \rceil}$, for a suitable constant c. Apply the map $\phi = \psi \circ \chi$ to f, g. Now, $\mathcal{L}(\phi(f)), \mathcal{L}(\phi(g)) \leq c\,d\,(p+q)^k \max\{\tau, \sigma\}$. By prop. 2, the evaluation costs $\widetilde{\mathcal{O}}_B(q(p+q)^{k+1}d\max\{\tau, \sigma\})$. □

We obtain the following, for bivariate $f, g \in (\mathbb{Z}[y])[x]$, such that $\mathtt{dg}_x(f) = p$, $\mathtt{dg}_x(g) = q$, $\mathtt{dg}_y(f), \mathtt{dg}_y(g) \leq d$.

Corollary 11. We compute $\mathbf{SR}(f, g)$ in $\widetilde{\mathcal{O}}_B(pq(p+q)^2 d\tau)$. For any polynomial $\mathbf{SR}_j(f, g)$ in $\mathbf{SR}(f, g)$, $\mathtt{dg}_x(\mathbf{SR}_j(f, g)) = \mathcal{O}(\max\{p, q\})$, $\mathtt{dg}_y(\mathbf{SR}_j(f, g)) = \mathcal{O}(\max\{p, q\}d)$, and also $\mathcal{L}(\mathbf{SR}_j(f, g)) = \mathcal{O}(\max\{p, q\}\tau)$.

Algorithm 1: SIGN_AT(F, α, β)

Input: $F \in \mathbb{Z}[x, y], \alpha \cong (A, [\mathsf{a}_1, \mathsf{a}_2]), \beta \cong (B, [\mathsf{b}_1, \mathsf{b}_2])$
Output: $\text{sign}(F(\alpha, \beta))$
1 compute $\mathbf{SRQ}_x(A, F)$
2 $L_1 \leftarrow \mathbf{SR}_x(A, F\,;\, \mathsf{a}_1)$, $V_1 \leftarrow \emptyset$
3 **foreach** $f \in L_1$ **do** $V_1 \leftarrow \text{ADD}(V_1,\ \text{SIGN\_AT}(f, \beta))$
4 $L_2 \leftarrow \mathbf{SR}_x(A, F\,;\, \mathsf{a}_2)$, $V_2 \leftarrow \emptyset$
5 **foreach** $f \in L_2$ **do** $V_2 \leftarrow \text{ADD}(V_2,\ \text{SIGN\_AT}(f, \beta))$
6 RETURN $(\text{VAR}(V_1) - \text{VAR}(V_2)) \cdot \text{sign}(A'(\alpha))$

Corollary 12. We compute $\mathbf{SRQ}(f, g)$, any polynomial in $\mathbf{SR}(f, g)$, and $\mathtt{res}(f, g)$ in $\widetilde{\mathcal{O}}_B(pq \max\{p, q\} d\tau)$.

Corollary 13. We can compute $\mathbf{SR}(f, g\,;\, \mathsf{a})$, where $\mathsf{a} \in \mathbb{Q} \cup \{\infty\}$ and $\mathcal{L}(\mathsf{a}) = \sigma$, in $\widetilde{\mathcal{O}}_B(pq \max\{p, q\} d \max\{\tau, \sigma\})$. For the polynomials $\mathbf{SR}_j(f, g\,;\, \mathsf{a}) \in \mathbb{Z}[y]$, except for f, g, it holds $\mathsf{dg}_y(\mathbf{SR}_j(f, g\,;\, \mathsf{a})) = \mathcal{O}((p + q)d)$ and $\mathcal{L}(\mathbf{SR}_j(f, g\,;\, \mathsf{a})) = \mathcal{O}(\max\{p, q\}\tau + \min\{p, q\}\sigma)$.

We now reduce the computation of the sign of $F \in \mathbb{Z}[x, y]$ over $(\alpha, \beta) \in \mathbb{R}^2_{alg}$ to that over several points in $\mathbb{Q}^2$. Let $\mathsf{dg}_x(F) = \mathsf{dg}_y(F) = n_1$, $\mathcal{L}(F) = \sigma$ and $\alpha \cong (A, [\mathsf{a}_1, \mathsf{a}_2])$, $\beta \cong (B, [\mathsf{b}_1, \mathsf{b}_2])$, where $A, B \in \mathbb{Z}[X]$, $\mathsf{dg}(A) = \mathsf{dg}(B) = n_2$, $\mathcal{L}(A) = \mathcal{L}(B) = \sigma$. We assume $n_1 \leq n_2$, which is relevant below. The algorithm is alg. 1, see [31], and generalizes the univariate case, e.g. [15, 38]. For A, resp. B, we assume that we know their values on $\mathsf{a}_1, \mathsf{a}_2$, resp. $\mathsf{b}_1, \mathsf{b}_2$.

Theorem 14. We compute the sign of polynomial $F(x, y)$ over α, β in $\widetilde{\mathcal{O}}_B(n_1^2\, n_2^3\, \sigma)$.

Proof. First, we compute $\mathbf{SRQ}_x(A, F)$, in $\widetilde{\mathcal{O}}_B(n_1^2 n_2^2 \sigma)$ (cor. 12), so as to evaluate $\mathbf{SR}(A, F)$ on the endpoints of α.

We compute $\mathbf{SR}_x(A, F; \mathsf{a}_1)$. The first polynomial in the sequence is A and notice that we already know its value on a_1. This computation costs $\widetilde{\mathcal{O}}_B(n_1^2\, n_2^3\, \sigma)$ by cor. 13 with $q = n_1$, $p = n_2$, $d = n_1$, $\tau = \sigma$, and $\sigma = n_2\sigma$, where the latter corresponds to the bitsize of the endpoints. After the evaluation we obtain a list L_1, which contains $\mathcal{O}(n_1)$ polynomials, say $f \in \mathbb{Z}[y]$, such that $\mathsf{dg}(f) = \mathcal{O}(n_1 n_2)$. To bound the bitsize, notice that the polynomials in $\mathbf{SR}(f, g)$ are of degrees $\mathcal{O}(n_1)$ w.r.t. x and of bitsize $\mathcal{O}(n_2\sigma)$. After we evaluate on a_1, $\mathcal{L}(f) = \mathcal{O}(n_1 n_2 \sigma)$.

For each $f \in L_1$ we compute its sign over β and count the sign variations. We could apply directly cor. 5, but we can do better. If $\mathsf{dg}(f) \geq n_2$ then $\mathbf{SR}(B, f) = (B, f, -B, g = -\,\mathsf{prem}(f, -B), \dots)$. We start the evaluations at g: it is computed in $\widetilde{\mathcal{O}}_B(n_1^2 n_2^3 \sigma)$ (prop. 1), $\mathsf{dg}(g) = \mathcal{O}(n_2)$ and $\mathcal{L}(g) = \mathcal{O}(n_1 n_2 \sigma)$. Thus, we evaluate $\mathbf{SR}(-B, g; \mathsf{a}_1)$ in $\widetilde{\mathcal{O}}_B(n_1 n_2^3 \sigma)$, by cor. 5, with $p = q = n_2$, $\tau_f = \sigma$, $\tau = n_1 n_2 \sigma$. If $\mathsf{dg}(f) < n_2$ the complexity is dominated. Since we perform $\mathcal{O}(n_1)$ such evaluations, all of them cost $\widetilde{\mathcal{O}}_B(n_1^2 n_2^3 \sigma)$.

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

Algorithm 2: GRID(F, G)

Input: $F, G \in \mathbb{Z}[x, y]$
Output: The real solutions of $F = G = 0$

```
1 R_x ← res_y(F, G)
2 L_x, M_x ← SOLVE(R_x)
3 R_y ← res_x(F, G)
4 L_y, M_y ← SOLVE(R_y)
5 Q ← ∅
6 foreach α ∈ L_x do
7     foreach β ∈ L_y do
8         if SIGN_AT(F, α, β) = 0 ∧ SIGN_AT(G, α, β) = 0 then Q ← ADD(Q, {α, β})
9 RETURN Q
```

4. Bivariate real solving

Let $F, G \in \mathbb{Z}[x, y]$, $\mathtt{dg}(F) = \mathtt{dg}(G) = n$ and $\mathcal{L}(F) = \mathcal{L}(G) = \sigma$. 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 the problem of real solving the system $F = G = 0$. The main idea is to project the roots on their x- and y-coordinates. The difference between the algorithms is the way they match coordinates.

4.1. The GRID *algorithm*

Algorithm GRID, is straightforward, see also [14, 36]. The pseudo-code is in alg. 2. We compute the x- and y-coordinates of the real solutions by solving resultants $\mathtt{res}_x(F, G)$, $\mathtt{res}_y(F, G)$. We match them using the algorithm SIGN_AT (th. 14) by testing all rectangles in this grid.

To the best of our knowledge, this is the first time that the algorithm's complexity is studied. Its simplicity makes it attractive; however, SIGN_AT (alg. 1) is very costly. 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, such as bounding the number of roots, e.g. Mixed Volume, or counting the roots with given abscissa by lem. 21.

Theorem 15. Isolating all real roots of system $F = G = 0$ using GRID has complexity $\widetilde{\mathcal{O}}_B(n^{14} + n^{13}\sigma)$, provided $\sigma = \mathcal{O}(n^3)$; or in $\widetilde{\mathcal{O}}_B(N^{14})$, where $N = \max\{n, \sigma\}$.

Proof. We compute resultant R_x of F, G w.r.t. y (line 1 in alg. 2). The complexity is $\widetilde{\mathcal{O}}_B(n^4\sigma)$, using cor. 12, with $p = q = d = n$ and $\tau = \sigma$. Notice that $\mathtt{dg}(R_x) = \mathcal{O}(n^2)$, $\mathcal{L}(R_x) = \mathcal{O}(n\,\sigma)$. We isolate its real roots in $\widetilde{\mathcal{O}}_B(n^{12} + n^{10}\sigma^2)$ (prop. 4) and store them in L_x. This complexity shall be dominated. We do the same for the y-axis (lines 3 and 4 in alg. 2) and store the roots in L_y.

The representation of the algebraic numbers contains the square-free part of R_x or R_y, which has the bitsize $\mathcal{O}(n^2+n\,\sigma)$ [3, 15]. The isolating intervals have endpoints of bitsize $\mathcal{O}(n^4+n^3\,\sigma)$. Let r_x, r_y be the number of real roots of the corresponding resultant, both in $\mathcal{O}(n^2)$. For every pair of algebraic numbers from L_x and L_y, we test whether F, G vanish using SIGN_AT (th. 14 and alg. 1). Each test costs $\widetilde{\mathcal{O}}_B(n^{10}+n^9\sigma)$ and we perform $r_x\, r_y = O(n^4)$ of them. □

We now examine the multiplicity of a root (α, β) of the system. Refer to [4, sec.II.6] for its definition as the exponent of factor $(\beta x - \alpha y)$ in the resultant of the (homogenized) polynomials, under certain assumptions. Previous work includes [16, 32, 37]. Our algorithm reduces to bivariate sign determination and does not require bivariate factorization. The sum of multiplicities of all roots (α, β_j) equals the multiplicity of $x = \alpha$ in the respective resultant. We apply a shear transform so as to ensure that different roots project to different points on the x-axis.

4.1.1. Deterministic shear

We determine an adequate (horizontal) shear such that

$$R_t(x) = \mathtt{res}_y\left(F(x+ty, y), G(x+ty, y)\right), \tag{1}$$

has simple roots corresponding to the projections of the common roots of the system $F(x, y) = G(x, y) = 0$, when $t \mapsto t_0 \in \mathbb{Z}$, and the degree of the polynomials remains the same. Notice that this shear does not affect inherently multiple roots, which exist independently of the reference frame. $R_{red} \in (\mathbb{Z}[t])[x]$ is the squarefree part of the resultant, as an element of UFD $(\mathbb{Z}[t])[x]$, and its discriminant, with respect to x, is $\Delta \in \mathbb{Z}[t]$. Then t_0 must be such that $\Delta(t_0) \neq 0$.

Lemma 16. Computing $t_0 \in \mathbb{Z}$, such that the corresponding shear is sufficiently generic, has complexity $\widetilde{\mathcal{O}}_B(n^{10}+n^9\sigma)$.

Proof. Suppose t_0 is such that the degree does not change. It suffices to find, among n^4 integer numbers, one that does not make Δ vanish; note that all candidate values are of bitsize $\mathcal{O}(\log n)$.

We perform the substitution $(x, y) \mapsto (x+ty, y)$ to F and G and compute the resultant w.r.t. y in $\widetilde{\mathcal{O}}_B(n^5\sigma)$, which lies in $\mathbb{Z}[t, x]$, of degree $\mathcal{O}(n^2)$ and bitsize $\widetilde{\mathcal{O}}(d\sigma)$ (prop. 8). We consider this polynomial as univariate in x and compute its square-free part, and then the discriminant of its square-free part. Both operations cost $\widetilde{\mathcal{O}}_B(n^{10}+n^9\sigma)$ and the discriminant is a polynomial in $\mathbb{Z}[t]$ of degree $\mathcal{O}(n^4)$ and bitsize $\widetilde{\mathcal{O}}(d^4+d^3\sigma)$ (cor. 12).

We can evaluate the discriminant over all the first n^4 positive integers, in $\widetilde{\mathcal{O}}_B(n^8+n^3\sigma)$, using the multipoint evaluation algorithm, see, e.g., [34], Among these integers, there is at least one that is not a root of the discriminant. □

The idea here is to use explicit candidate values of t_0 right from the start. In practice, the above complexity becomes $\widetilde{\mathcal{O}}_B(n^5\sigma)$, because a constant number of tries or a random value will typically suffice. For an alternative approach, see [17], and [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 17. Consider the setting of th. 15. Having isolated all real roots of $F = G = 0$, it is possible to determine their multiplicities in $\widetilde{\mathcal{O}}_B(n^{12}+n^{11}\sigma+n^{10}\sigma^2)$.

Algorithm 3: M_RUR (F, G)

Input: $F, G \in \mathbb{Z}[X, Y]$ in generic position
Output: The real solutions of the system $F = G = 0$

$\mathbf{SR} \leftarrow \mathbf{SR}_y(F, G)$
/* Projections and real solving with multiplicities */
$R_x \leftarrow \mathtt{res}_y(F, G)$
$P_x, M_x \leftarrow$ SOLVE(R_x)
$R_y \leftarrow \mathtt{res}_x(F, G)$
$P_y, M_y \leftarrow$ SOLVE(R_y)
$I \leftarrow$ INTERMEDIATE_POINTS(P_y)
/* Factorization of R_x according to sr */
$K \leftarrow$ COMPUTE_K$(\mathbf{SR}, P_x)$
$Q \leftarrow \emptyset$
/* Matching the solutions */
foreach $\alpha \in P_x$ **do**
  $\beta \leftarrow$ FIND(α, K, P_y, I)
  $Q \leftarrow$ ADD$(Q, \{\alpha, \beta\})$
RETURN Q

Proof. By the previous lemma, $t \in \mathbb{Z}$ is determined, with $\mathcal{L}(t) = \mathcal{O}(\log n)$, in $\widetilde{\mathcal{O}}_B(n^{10} + n^9\sigma)$. Using this value, we isolate all the real roots of $R_t(x)$, defined in (1), and determine their multiplicities in $\widetilde{\mathcal{O}}_B(n^{12} + n^{10}\sigma^2)$ (prop. 4). Let $\rho_j \simeq (R_t(x), [r_j, r'_j])$ be the real roots, for $j = 0, \ldots, r-1$.

By assumption, we have already isolated the roots of the system, denoted by $(\alpha_i, \beta_i) \in [a_i, a'_i] \times [b_i, b'_i]$, where $a_i, a'_i, b_i, b'_i \in \mathbb{Q}$ for $i = 0, \ldots, r-1$. It remains to match each pair (α_i, β_i) to a unique ρ_j by determining function $\phi : \{0, \ldots, r-1\} \to \{0, \ldots, r-1\}$, such that $\phi(i) = j$ iff $(\rho_j, \beta_i) \in \mathbb{R}^2_{alg}$ is a root of the sheared system and $\alpha_i = \rho_j + t\beta_i$.

Let $[c_i, c'_i] = [a_i, a'_i] - t[b_i, b'_i] \in \mathbb{Q}^2$. These intervals may be overlapping. Since the endpoints have bitsize $\mathcal{O}(n^4 + n^3\sigma)$, the intervals $[c_i, c'_i]$ are sorted in $\widetilde{\mathcal{O}}_B(n^6 + n^5\sigma)$. The same complexity bounds the operation of merging this interval list with the list of intervals $[r_j, r'_j]$. If there exist more than one $[c_i, c'_i]$ overlapping with some $[r_j, r'_j]$, some subdivision steps are required so that the intervals reach the bitsize of s_j, where 2^{s_j} bounds the separation distance associated to the j-th root. By prop. 6, $\sum_i s_i = \mathcal{O}(n^4 + n^3\sigma)$.

Our analysis resembles that of [15] for proving prop. 4. The total number of steps is $\mathcal{O}(\sum_i s_i) = \mathcal{O}(n^4 + n^3\sigma)$, each requiring an evaluation of $R(x)$ over an endpoint of size $\leq s_i$. This evaluation costs $\widetilde{\mathcal{O}}_B(n^4 s_i)$, leading to an overall cost of $\widetilde{\mathcal{O}}_B(n^8 + n^7\sigma)$ per level of the tree of subdivisions. Hence, the overall complexity is bounded by $\widetilde{\mathcal{O}}_B(n^{12} + n^{11}\sigma + n^{10}\sigma^2)$. □

4.2. The M_RUR *algorithm*

M_RUR 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 18. [3, 16] Let F, G be co-prime polynomials, in generic position. If $\mathbf{SR}_j(x,y) = \mathbf{sr}_j(x)y^j + \mathbf{sr}_{j,j-1}(x)y^{j-1} + \cdots + \mathbf{sr}_{j,0}(x)$, and (α,β) is a real solution of the system $F = G = 0$, then there exists k, such that $\mathbf{sr}_0(\alpha) = \cdots = \mathbf{sr}_{k-1}(\alpha) = 0$, $\mathbf{sr}_k(\alpha) \neq 0$ and $\beta = -\frac{1}{k}\frac{\mathbf{sr}_{k,k-1}(\alpha)}{\mathbf{sr}_k(\alpha)}$.

This expresses the ordinate of a solution in a Rational Univariate Representation (RUR) of the abscissa. The RUR applies to multivariate algebraic systems [3, 6, 29, 30] by generalizing the primitive-element method by Kronecker. Here we adapt it to small-dimensional systems.

Our algorithm is similar to [16, 17]. However, their algorithm computes only a RUR using prop. 18, so the representation of the ordinates remains implicit. Often, this representation is not sufficient (we can always compute the minimal polynomial of the roots, but this is highly inefficient). We modified the algorithm [14], so that the output includes isolating rectangles, hence the name modified-RUR (M_RUR). The most important difference with [16] is that they represent algebraic numbers by Thom's encoding while we use isolating intervals, which were thought of having high theoretical complexity.

The pseudo-code of M_RUR is in alg. 3. We project on the x and the y-axis; for each real solution on the x-axis we compute its ordinate using prop. 18. First we compute the sequence $\mathbf{SR}(F,G)$ w.r.t. y in $\widetilde{\mathcal{O}}_B(n^5\,\sigma)$ (cor. 11).

Projection. This is similar to GRID. The complexity is dominated by real solving the resultants, i.e. $\widetilde{\mathcal{O}}_B(n^{12} + n^{10}\,\sigma^2)$. Let α_i, resp. β_j, be the real root coordinates. We compute rationals q_j between the β_j's in $\widetilde{\mathcal{O}}_B(n^5\sigma)$, viz. INTERMEDIATE_POINTS(P_y); the q_j have aggregate bitsize $\mathcal{O}(n^3\,\sigma)$ (lem. 6):

$$q_0 < \beta_1 < q_1 < \beta_2 < \cdots < \beta_{\ell-1} < q_{\ell-1} < \beta_\ell < q_\ell, \tag{2}$$

where $\ell \leq 2\,n^2$. Every β_j corresponds to a unique α_i. The multiplicity of α_i as a root of R_x is the multiplicity of a real solution of the system, that has it as abscissa.

Sub-algorithm COMPUTE_K. In order to apply prop. 18, for every α_i we must compute $k \in \mathbb{N}^*$ such that the assumptions of the theorem are fulfilled; this is possible by genericity. We follow [16, 25] and define recursively polynomials $\Gamma_j(x)$: Let $\Phi_0(x) = \frac{\mathbf{sr}_0(x)}{\mathsf{gcd}(\mathbf{sr}_0(x),\mathbf{sr}_0'(x))}$, $\Phi_j(x) = \mathsf{gcd}(\Phi_{j-1}(x), \mathbf{sr}_j(x))$, and $\Gamma_j = \frac{\Phi_{j-1}(x)}{\Phi_j(x)}$, for $j > 0$. Now $\mathbf{sr}_i(x) \in \mathbb{Z}[x]$ is the principal subresultant coefficient of $\mathbf{SR}_i \in (\mathbb{Z}[x])[y]$, and $\Phi_0(x)$ is the square-free part of $R_x = \mathbf{sr}_0(x)$. By construction, $\Phi_0(x) = \prod_j \Gamma_j(x)$ and $\mathsf{gcd}(\Gamma_j, \Gamma_i) = 1$, if $j \neq i$. Hence every α_i is a root of a unique Γ_j and the latter switches sign at the interval's endpoints. Then, $\mathbf{sr}_0(\alpha) = \mathbf{sr}_1(\alpha) = 0, \ldots, \mathbf{sr}_j(\alpha) = 0$, $\mathbf{sr}_{j+1}(\alpha) \neq 0$; thus $k = j+1$.

It holds that $\mathsf{dg}(\Phi_0) = \mathcal{O}(n^2)$ and $\mathcal{L}(\Phi_0) = \mathcal{O}(n^2 + n\,\sigma)$. Moreover, $\sum_j \mathsf{dg}(\Gamma_j) = \sum_j \delta_j = \mathcal{O}(n^2)$ and, by Mignotte's bound [21], $\mathcal{L}(\Gamma_j) = \mathcal{O}(n^2 + n\sigma)$. To compute the factorization $\Phi_0(x) = \prod_j \Gamma_j(x)$ as a product of the $\mathbf{sr}_j(x)$, we perform $\mathcal{O}(n)$ gcd computations of polynomials of degree $\mathcal{O}(n^2)$ and bitsize $\widetilde{\mathcal{O}}(n^2 + n\sigma)$. Each gcd computation costs $\widetilde{\mathcal{O}}_B(n^6 + n^5\,\sigma)$ (prop. 1) and thus the overall cost is $\widetilde{\mathcal{O}}_B(n^7 + n^6\,\sigma)$.

We compute the sign of the Γ_j over all the $O(n^2)$ isolating endpoints of the α_i, which have aggregate bitsize $\mathcal{O}(n^4 + n^3\,\sigma)$ (lem. 6) in $\widetilde{\mathcal{O}}_B(\delta_j n^4 + \delta_j n^3\sigma + \delta_j^2(n^4 + n^3\sigma))$, using Horner's rule. Summing over all δ_j, the complexity is $\widetilde{\mathcal{O}}_B(n^8 + n^7\sigma)$. Thus the overall complexity is $\widetilde{\mathcal{O}}_B(n^9 + n^8\,\sigma)$.

Matching and algorithm FIND. The process takes a real root of R_x and computes the ordinate β of the corresponding root of the system. For some real root α of R_x we represent the ordinate $A(\alpha) = -\frac{1}{k}\frac{\mathbf{sr}_{k,k-1}(\alpha)}{\mathbf{sr}_k(\alpha)} = \frac{A_1(\alpha)}{A_2(\alpha)}$. The generic position assumption guarantees that there is a unique β_j, in P_y, such that $\beta_j = A(\alpha)$, where $1 \leq j \leq \ell$. In order to compute j we use (2): $q_j < A(\alpha) = \frac{A_1(\alpha)}{A_2(\alpha)} = \beta_j < q_{j+1}$. Thus j can be computed by binary search in $\mathcal{O}(\lg \ell) = \mathcal{O}(\lg n)$ comparisons of $A(\alpha)$ with the q_j. This is equivalent to computing the sign of $B_j(X) = A_1(X) - q_j\, A_2(X)$ over α by executing $\mathcal{O}(\lg n)$ times, SIGN_AT(B_j, α).

Now, $\mathcal{L}(q_j) = \mathcal{O}(n^4 + n^3\sigma)$ and $\mathsf{dg}(A_1) = \mathsf{dg}(\mathbf{sr}_{k,k-1}) = \mathcal{O}(n^2)$, $\mathsf{dg}(A_2) = \mathsf{dg}(\mathbf{sr}_k) = \mathcal{O}(n^2)$, $\mathcal{L}(A_1) = \mathcal{O}(n\,\sigma)$, $\mathcal{L}(A_2) = \mathcal{O}(n\,\sigma)$. Thus $\mathsf{dg}(B_j) = \mathcal{O}(n^2)$ and $\mathcal{L}(B_j) = \mathcal{O}(n^4 + n^3\,\sigma)$. We conclude that SIGN_AT(B_j, α) and FIND have complexity $\widetilde{\mathcal{O}}_B(n^8 + n^7\sigma)$ (cor. 5). As for the overall complexity of the loop (Lines 9-11) the complexity is $\widetilde{\mathcal{O}}_B(n^{10} + n^9\sigma)$, since it is executed $\mathcal{O}(n^2)$ times.

Theorem 19. We isolate all real roots of $F = G = 0$, if F, G are in generic position, by M_RUR in $\widetilde{\mathcal{O}}_B(n^{12} + n^{10}\sigma^2)$; or in $\widetilde{\mathcal{O}}_B(N^{12})$, where $N = \max\{n, \sigma\}$.

The generic position assumption is without loss of generality since we can always put the system in such position by applying a shear transform; see sec. 4.1.1 and also [3, 16]. The bitsize of polynomials of the (sheared) system becomes $\widetilde{\mathcal{O}}(n + \sigma)$ [16] and does not change the bound of th. 19. However, now is raised the problem of expressing the real roots in the original coordinate system (see the proof of th. 17).

4.3. The G_RUR *algorithm*

In this section we present an algorithm that uses some ideas from M_RUR but also relies on GCD computations of polynomials with coefficients in an extension field to achieve efficiency (hence the name G_RUR). The pseudo-code of G_RUR is in alg. 4. For GCD computations with polynomials with coefficients in an extension field we use the algorithm, and the MAPLE implementation, of van Hoeij and Monagan [33].

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

For each x-coordinate, say α, we compute the square-free part of $F(\alpha, y)$ and $G(\alpha, y)$, say $\bar{F}$ and $\bar{G}$. The complexity is that of computing the gcd with the derivative. In [33] the cost is $\widetilde{\mathcal{O}}_B(mMND + mN^2D^2 + m^2kD)$, where M is the bitsize of the largest coefficient, N is the degree of the largest polynomial, D is the degree of the extension, k is the degree of the gcd, and m is the number of primes needed. This bound does not assume fast multiplication algorithms, thus, under this assumption, it becomes $\widetilde{\mathcal{O}}_B(mMND + mND + mkD)$.

In our case $M = \mathcal{O}(\sigma)$, $N = \mathcal{O}(n)$, $D = \mathcal{O}(n^2)$, $k = \mathcal{O}(n)$, and $m = \mathcal{O}(n\sigma)$. The cost is $\widetilde{\mathcal{O}}_B(n^4\sigma^2)$ and since we repeat it $\mathcal{O}(n^2)$ times, the overall cost is $\widetilde{\mathcal{O}}_B(n^6\sigma^2)$. Notice the bitsize of the result is $\widetilde{\mathcal{O}}_B(n + \sigma)$ [3].

Now for each α, we compute $H = \gcd(\bar{F}, \bar{G})$. We have $M = \mathcal{O}(n+\sigma)$, $N = \mathcal{O}(n)$, $D = \mathcal{O}(n^2)$, $k = \mathcal{O}(n)$, and $m = \mathcal{O}(n^2 + n\sigma)$, so the cost of each operation is $\widetilde{\mathcal{O}}_B(n^6 + n^4\sigma^2)$ and overall $\widetilde{\mathcal{O}}_B(n^8 + n^6\sigma^2)$. The size of m comes from Mignotte's bound [21]. H is a square-free polynomial in $(\mathbb{Z}[\alpha])[y]$, of degree $\mathcal{O}(n)$ and bitsize $\mathcal{O}(n^2 + n\sigma)$, whose real

Algorithm 4: G_RUR (F, G)

Input: $F, G \in \mathbb{Z}[x, y]$
Output: The real solutions of the system $F = G = 0$
/* Projections and real solving with multiplicities */
$R_x \leftarrow \mathtt{res}_y(F, G)$
$P_x, M_x \leftarrow$ SOLVE(R_x)
$R_y \leftarrow \mathtt{res}_x(F, G)$
$P_y, M_y \leftarrow$ SOLVE(R_y)
/* I contains the rationals $q_1 < q_2 < \cdots < q_{|I|}$ */
$I \leftarrow$ INTERMEDIATE_POINTS(P_y)
$Q \longleftarrow \emptyset$
foreach $\alpha \in P_x$ **do**
  $\bar{F} \longleftarrow$ SQUAREFREEPART$(F(\alpha, y))$
  $\bar{G} \longleftarrow$ SQUAREFREEPART$(G(\alpha, y))$
  $H \longleftarrow$ GCD$(\bar{F}, \bar{G}) \in (\mathbb{Z}[\alpha])[y]$
  for $j \leftarrow 1$ **to** $|I| - 1$ **do**
    if $H(\alpha, q_j) \cdot H(\alpha, q_{j+1}) < 0$ **then**
      /* $P_y[j]$ indicates the $j-$th element of P_y */
      $Q \leftarrow$ ADD$(Q, \{\alpha, P_y[j]\})$
RETURN Q

roots correspond to the real solutions of the system with abscissa α. The crux of the method is that H changes sign only over the intervals that contain its real roots. To check these signs, it suffices to substitute y in H by the intermediate points, thus obtaining a polynomial in $\mathbb{Z}[\alpha]$, of degree $\mathcal{O}(n)$ and bitsize $\mathcal{O}(n^2 + n\sigma + ns_j)$, where s_j is the bitsize of the j-th intermediate point.

Now, we consider this polynomial in $\mathbb{Z}[x]$ and evaluate it over α. Using cor. 5 with $p = n^2$, $\tau_f = n^2 + n\sigma$, $q = n$, and $\tau_g = n^2 + n\sigma + ns_j$, this costs $\widetilde{\mathcal{O}}_B(n^6 + n^5\sigma + n^4 s_j)$. Summing over $\mathcal{O}(n^2)$ points and using lem. 6, we obtain $\widetilde{\mathcal{O}}_B(n^8 + n^7\sigma)$. Thus, the overall complexity is $\widetilde{\mathcal{O}}_B(n^{10} + n^9\sigma)$.

Theorem 20. We can isolate the real roots of the system $F = G = 0$, using G_RUR in $\widetilde{\mathcal{O}}_B(n^{12} + n^{10}\sigma^2)$; or $\widetilde{\mathcal{O}}_B(N^{12})$, where $N = \max\{n, \sigma\}$.

5. Applications

5.1. Real root counting

Let $F \in \mathbb{Z}[x, y]$, such that $\mathsf{dg}_x(F) = \mathsf{dg}_y(F) = n_1$ and $\mathcal{L}(F) = \sigma$. Let $\alpha, \beta \in \mathbb{R}_{alg}$, such that $\alpha \cong (A, [\mathsf{a}_1, \mathsf{a}_2])$ and $\beta \cong (B, [\mathsf{b}_1, \mathsf{b}_2])$, where $\mathsf{dg}(A), \mathsf{dg}(B) = n_2, \mathcal{L}(A), \mathcal{L}(B) \leq \tau$ and $\mathsf{c} \in \mathbb{Q}$, such that $\mathcal{L}(\mathsf{c}) = \lambda$. Moreover, assume that $n_1^2 = \mathcal{O}(n_2)$, as is the case in applications. We want to count the number of real roots of $\bar{F} = F(\alpha, y) \in (\mathbb{Z}(\alpha))[y]$ in $(-\infty, +\infty)$, in $(\mathsf{c}, +\infty)$ and in $(\beta, +\infty)$. We may assume that the leading coefficient of $\bar{F}$ is nonzero. This is w.l.o.g. since we can easily check it, and/or we can use the good specialization properties of the subresultants [16, 18, 20].

Using Sturm's theorem, e.g. [3, 38], the number of real roots of $\bar{F}$ is $\mathtt{VAR}(\mathbf{SR}(\bar{F}, \bar{F}_y; -\infty)) - \mathtt{VAR}(\mathbf{SR}(\bar{F}, \bar{F}_y; +\infty))$. Hence, we have to compute the sequence $\mathbf{SR}(\bar{F}, \bar{F}_y)$ w.r.t. y, and evaluate it on $\pm\infty$ or, equivalently, to compute the signs of the principal subresultant coefficients, which lie in $\mathbb{Z}(\alpha)$. This procedure is equivalent, due to the good specialization properties of subresultants [3, 18], to computing the principal subresultant coefficients of $\mathbf{SR}(F, F_y)$, which are polynomials in $\mathbb{Z}[x]$, and to evaluate them over α. In other words, the good specialization properties assure us that we can compute a nominal sequence by considering the bivariate polynomials, and then perform the substitution $x = \alpha$.

The sequence $\mathbf{sr}$ of the principal subresultant coefficients can be computed in $\widetilde{\mathcal{O}}_B(n_1^4\sigma)$, using cor. 12 with $p = q = d = n_1$, and $\tau = \sigma$. Now, $\mathbf{sr}$ contains $\mathcal{O}(n_1)$ polynomials in $\mathbb{Z}[x]$, each of degree $\mathcal{O}(n_1^2)$ and bitsize $\mathcal{O}(n_1\sigma)$. We compute the sign of each one evaluated over α in

$$\widetilde{\mathcal{O}}_B(n_1^2 n_2 \max\{\tau, n_1\sigma\} + n_2 \min\{n_1^2, n_2\}^2\tau)$$

using cor. 5 with $p = n_2$, $q = n_1^2$, $\tau_f = \tau$, and $\tau_g = n_1\sigma$. This proves the following:

Lemma 21. We count the number of real roots of $\bar{F} = F(\alpha, y)$ in $\widetilde{\mathcal{O}}_B(n_1^4 n_2\sigma + n_1^5 n_2\tau)$.

In order to count the real roots of $\bar{F}$ in $(\beta, +\infty)$, we use again Sturm's theorem. The complexity of the computation is dominated by the cost of computing $\mathtt{VAR}(\mathbf{SR}(\bar{F}, \bar{F}_y; \beta))$, which is equivalent to computing $\mathbf{SR}(F, F_y)$ w.r.t. to y, which contains bivariate polynomials, and to compute their signs over (α, β). The cost of computing $\mathbf{SR}(F, F_y)$ is $\widetilde{\mathcal{O}}_B(n_1^5\sigma)$ using cor. 11 with $p = q = d = n_1$, and $\tau = \sigma$. The sequence contains $\mathcal{O}(n_1)$ polynomials in $\mathbb{Z}[x, y]$ of degrees $\mathcal{O}(n_1)$ and $\mathcal{O}(n_1^2)$, w.r.t. x and y respectively, and bitsize $\mathcal{O}(n_1\sigma)$. We compute the sign of each over (α, β) in $\widetilde{\mathcal{O}}_B(n_1^4 n_2^3 \max\{n_1\sigma, \tau\})$ (th. 14). This proves the following:

Lemma 22. We count the number of real roots of $\bar{F}$ in $(\beta, +\infty)$ in $\widetilde{\mathcal{O}}_B(n_1^5 n_2^3 \max\{n_1\sigma, \tau\})$.

By a more involved analysis, taking into account the difference in the degrees of the bivariate polynomials, we can gain a factor. We omit it for reasons of simplicity. Finally, in order to count the real roots of $\bar{F}$ in $(\mathsf{c}, +\infty)$, it suffices to evaluate the sequence $\mathbf{SR}(F, F_y)$ w.r.t. y on c, thus obtaining polynomials in $\mathbb{Z}[x]$, and compute their signs over α.

The cost of the evaluation $\mathbf{SR}(F, F_y; \mathsf{c})$ is $\widetilde{\mathcal{O}}_B(n_1^4 \max\{\sigma, \lambda\})$, using cor. 13 with $p = q = d = n_1$, $\tau = \sigma$ and $\sigma = \lambda$. The evaluated sequence contains $\mathcal{O}(n_1)$ polynomials in $\mathbb{Z}[x]$, of degree $\mathcal{O}(n_1^2)$ and bitsize $\mathcal{O}(n_1 \max\{\sigma, \lambda\})$. The sign of each one evaluated over α can be compute in

$$\widetilde{\mathcal{O}}_B(n_1^2 n_2 \max\{\tau, n_1\sigma, n_1\lambda\} + n_1^4 n_2\tau),$$

using cor. 5 with $p = n_2$, $q = n_1^2$, $\tau_f = \tau$ and $\tau_g = n_1 \max\{\sigma, \lambda\}$. This leads to the following:

Lemma 23. We count the number of real roots of $\bar{F}$ in $(\mathsf{c}, +\infty)$ in $\widetilde{\mathcal{O}}_B(n_1^4 n_2 \max\{n_1\tau, \sigma, \lambda\})$.

5.2. 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 σ. We wish to compute $(\alpha, \beta) \in \mathbb{R}^2_{alg}$ such that $P(\alpha, \beta) = Q(\alpha, \beta) = 0$ and also $A_i(\alpha, \beta) > 0$, $B_j(\alpha, \beta) < 0$ and $C_k(\alpha, \beta) = 0$, 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 24. There is an algorithm that solves the problem of ℓ simultaneous inequalities of degree $\leq n$ and bitsize $\leq \sigma$, in $\widetilde{\mathcal{O}}_B(\ell n^{12} + \ell n^{11}\sigma + n^{10}\sigma^2)$.

Proof. Initially we compute the isolating interval representation of the real roots of $P = Q = 0$ in $\widetilde{\mathcal{O}}_B(n^{12}+n^{10}\sigma^2)$, using GRUR_SOLVE. There are $\mathcal{O}(n^2)$ real solutions, which are represented in isolating interval representation, with polynomials of degrees $\mathcal{O}(n^2)$ and bitsize $\mathcal{O}(n^2 + n\sigma)$.

For each real solution, say (α, β), for each polynomial A_i, B_j, C_k we compute the signs of $\text{sign}(A_i(\alpha, \beta))$, $\text{sign}\,(B_i(\alpha, \beta))$ and $\text{sign}\,(C_i(\alpha, \beta))$. Each sign evaluation costs $\widetilde{\mathcal{O}}_B(n^{10} + n^9\sigma)$, using th. 14 with $n_1 = n$, $n_2 = n^2$ and $\sigma = n^2 + n\sigma$. In the worst case we need n^2 of them, hence, the cost for all sign evaluations is $\widetilde{\mathcal{O}}_B(\ell n^{12} + \ell\, n^{11}\,\sigma)$. □

5.3. The complexity of topology

In this section we consider the problem of computing the topology of a real plane algebraic curve, and improve upon its asymptotic complexity. The reader may refer to, e.g., [3, 16, 25], for the details of the algorithm.

We consider the curve in generic position (sec. 4.1.1), defined by $F \in \mathbb{Z}[x, y]$, such that $\mathsf{dg}(F) = n$ and $\mathcal{L}\,(F) = \sigma$. We compute the critical points of the curve, i.e. solve $F = F_y = 0$ in $\widetilde{\mathcal{O}}_B(n^{12} + n^{10}\sigma^2)$, where F_y is the derivative of F w.r.t y. Next, we compute the intermediate points on the x axis, in $\widetilde{\mathcal{O}}_B(n^4 + n^3\sigma)$ (lem. 6). 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 $\mathcal{O}(n\mathcal{L}\,(q_j))$. For this we use Sturm's theorem and th. 2 and the cost is $\widetilde{\mathcal{O}}_B(n^3\mathcal{L}\,(q_j))$. For all q_j's the cost is $\widetilde{\mathcal{O}}_B(n^7 + n^6\sigma)$.

For each critical point, say (α, β) 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 lem. 21, in $\widetilde{\mathcal{O}}_B(n^9 + n^8\sigma)$, where $n_1 = n$, $n_2 = n^2$, and $\tau = n^2 + n\sigma$. Since there are $\mathcal{O}(n^2)$ critical values, the overall cost of the step is $\widetilde{\mathcal{O}}_B(n^{11} + n^{10}\sigma)$.

Finally, we compute the number of branches that cross the line $x = \alpha$ and are above $y = \beta$ in $\widetilde{\mathcal{O}}_B(n^{13}+n^{12}\sigma)$, by lem. 22. Since there are $\mathcal{O}(n^2)$ critical points, the complexity is $\widetilde{\mathcal{O}}_B(n^{15} + n^{14}\sigma)$. 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 $\widetilde{\mathcal{O}}_B(n^{15} + n^{14}\sigma + n^{10}\sigma^2)$, or $\widetilde{\mathcal{O}}_B(N^{15})$, which is worse by a factor than [3].

We improve the complexity of the last step since M_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 β and perform univariate sign evaluations. This corresponds to computing the sign of $\mathcal{O}(n^2)$ polynomials of degree $\mathcal{O}(n^2)$ and bitsize $\mathcal{O}(n^4 + n^3\sigma)$, over all the α's [16]. Using lem. 7 for each polynomial the cost is $\widetilde{\mathcal{O}}_B(n^{10} + n^9\sigma)$, and since there are $\widetilde{\mathcal{O}}_B(n^2)$ of them, the total cost is $\widetilde{\mathcal{O}}_B(n^{12} + n^{11}\sigma)$.

Theorem 25. We compute the topology of a real plane algebraic curve, defined by a polynomial of degree n and bitsize σ, in $\widetilde{\mathcal{O}}_B(n^{12} + n^{11}\sigma + n^{10}\sigma^2)$, or $\widetilde{\mathcal{O}}_B(N^{12})$, where $N = \max\{n, \sigma\}$.

Thus the overall complexity of the algorithm improves the previously known bound by a factor of N^2. We assumed generic position, since we can apply a shear to achieve this, see sec. 4.1.

6. Implementation and Experiments

This section describes our open source MAPLE implementation [1] and illustrates its capabilities through comparative experiments. Refer to [10] for its usage and further details. Our design is object oriented and uses generic programming in view of transferring the implementation to `C++` in the future.

We provide algorithms for *signed* polynomial remainder sequences, real solving of univariate polynomials using Sturm's algorithm, computations with one and two real algebraic numbers, such as sign evaluation and comparison and, of course, solving bivariate systems.

6.1. Our solvers

The performance of all algorithms is averaged over 10 executions on a MAPLE 9.5 console using a 2GHz AMD64@3K+ processor with 1GB RAM. The polynomial systems tested are given in [10]: systems R_i, M_i, D_i are from [14], the C_i are from [17], and $W_i, i = 1, \ldots, 4$, follow from C_is after swapping x, y. The latter are of the form $f = \frac{\partial f}{\partial y} = 0$. For gcd computations in a (single) extension field, the package of [33] is used. The optimal algorithms for computing and evaluating polynomial remainder sequences have *not* yet been implemented.

Our main results are reported in table 1. G_RUR is the solver of choice since it is faster than GRID and M_RUR in 17 out of the 18 instances. However, this may not hold when the extension field is of high degree. G_RUR yields solutions in < 1 sec, apart from C_5. For total degree ≤ 8, G_RUR requires < 0.4 sec. On average, G_RUR is 7-11 times faster than GRID, and about 38 times faster than M_RUR. The inefficiency of M_RUR is due to the fact that it solves sheared systems which are dense and of increased bitsize; it also computes multiplicities. Finally, GRID reaches a stack limit with the default MAPLE stack size (8,192 KB) when solving C_5. Even when we multiplied stack size by 10, GRID did not terminate within 20 min.

Whenever we refer to the speedup we imply the fraction of runtimes. G_RUR can be up to 21.58 times faster than GRID with an average speedup of around 7.27 among the input systems (excluding C_5). With respect to M_RUR, G_RUR can be up to 275.74 times faster, with an average speedup of 38.01.

Filtering has been used. For this, two instances of isolating intervals are stored; one for filtering, another for exact computation. Probably, the most significant filtering technique is interval arithmetic. When computing the sign of a polynomial evaluated at a real algebraic number, the first attempt is via interval arithmetic, applied along with [1]. When this fails, and one wants to compare algebraic numbers or perform univariate SIGN_AT, then the gcd of two polynomials is computed.

Filtering helps most with M_RUR, especially when we compute multiplicities. With this solver, one more filter is used: the intervals of candidate x-solutions are refined by [1] so

[1] `www.di.uoa.gr/~erga/soft/SLV_index.html`

	phase of the algorithm	interval min	interval max	median	mean	std dev
GRID	projections	00.00	00.53	00.04	00.08	00.13
	univ. solving	02.05	99.75	07.08	26.77	35.88
	biv. solving	00.19	97.93	96.18	73.03	36.04
	sorting	00.00	01.13	00.06	00.12	00.26
MRUR	projection	00.00	00.75	00.06	00.14	00.23
	univ. solving	00.18	91.37	15.55	17.47	20.79
	StHa seq.	00.08	38.23	01.17	05.80	09.91
	inter. points	00.00	03.23	00.09	00.32	00.75
	filter x-cand	00.68	72.84	26.68	23.81	21.93
	compute K	00.09	34.37	02.04	07.06	10.21
	biv. solving	01.77	98.32	51.17	45.41	28.71
GRUR	projections	00.02	03.89	00.23	00.48	00.88
	univ. solving	07.99	99.37	39.83	41.68	25.52
	inter. points	00.02	03.81	00.54	01.11	01.28
	rational biv.	00.07	57.07	14.83	15.89	19.81
	$\mathbb{R}_{alg}$biv.	00.00	91.72	65.30	40.53	36.89
	sorting	00.00	01.50	00.22	00.32	00.43

(a) Statistics on SLV's sub-algorithms.

sys	**deg**		$\mathbb{R}_{\mathbf{alg}}$	**Avg Time (msec)**		
	f	**g**	**sols**	GRID	M_RUR	G_RUR
R_1	3	4	2	6	9	6
R_2	3	1	1	66	21	36
R_3	3	1	1	1	2	1
M_1	3	3	4	183	72	45
M_2	4	2	3	4	5	4
M_3	6	3	5	4,871	782	393
M_4	9	10	2	339	389	199
D_1	4	5	1	6	12	6
D_2	2	2	4	567	147	126
C_1	7	6	6	1,702	954	247
C_2	4	3	6	400	234	99
C_3	8	7	13	669	1,815	152
C_4	8	7	17	7,492	80,650	474
C_5	16	15	17	$> 20'$	60,832	6,367
W_1	7	6	9	3,406	2,115	393
W_2	4	3	5	1,008	283	193
W_3	8	7	13	1,769	2,333	230
W_4	8	7	17	5,783	77,207	709

(b) Performance of our solvers when computing multiplicities.

Fig. 1. Statistics

as to help the interval arithmetic filters inside FIND. If the above fails, we switch to exact computation via Sturm sequences, using the initial endpoints since they have smaller bitsize. In GRID's case, filtering provided an average speedup of 1.51, where C_5 has been excluded. With G_RUR, we have on average a speedup of 1.08. This is expected since G_RUR relies heavily on gcd's and factoring.

Fig. 1(a) shows the runtime breakdown corresponding to the various stages of each algorithm: `Projections` shows the time for computing resultants, `Univ.Solving` for solving them, and `Sorting` for sorting solutions. In GRID's and M_RUR's case, `biv.solving` corresponds to matching. In G_RUR's case, matching is divided between `rational biv` and $\mathbb{R}_{alg}$ `biv`; the first refers to when at least one of the co-ordinates is rational. `Inter.points` refers to computing intermediate points between resultant roots along the y-axis. `StHa seq` refers to computing 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 and does not take into account C_5. M_RUR spends 45-50% of its time in matching and 24-27% in filtering. G_RUR spends 55-80% of its time in matching, including gcd computations in an extension field.

In order to compute multiplicities, the initial systems were sheared whenever it was necessary, based on the algorithm presented in sec. 4.1.1. Overall results are shown in

Fig. 1(b). GRID's high complexity starts to become apparent. Overall, G_RUR is fastest and terminates within ≤ 1 sec. It can be up to 15.81 times faster than GRID with an average speedup of around 5.26. With respect to M_RUR, this time G_RUR can be up to 170.15 times faster, with an average speedup of around 18.77 among all input polynomial systems. M_RUR can be up to 6.23 times faster than GRID, yielding an average speedup of 1.71. A detailed table in [10] gives us the runtime decomposition of each algorithm in its major subroutines. Results are similar to sec. 6.1, except that G_RUR spends 68-80% of its time in matching, including gcd's. In absence of excessive factoring G_RUR spends significantly more time in bivariate solving.

6.2. Other software

FGB/RS [2] [30] performs exact real solving using Gröbner bases and RUR, through its MAPLE interface; additional tuning might offer 20-30% efficiency increase. Three SYNAPS [3] solvers have been tested: STURM is a naive implementation of GRID [14]; SUBDIV implements [23], using the Bernstein basis and `double` arithmetic. It needs an initial box and $[-10, 10] \times [-10, 10]$ was used. NEWMAC [24] is a general purpose solver based on eigenvectors using LAPACK, which computes all complex solutions.

MAPLE implementations: INSULATE implements [37] for computing the topology of real algebraic curves, and TOP implements [17]. Both packages were kindly provided by their authors. We tried to modify the packages so as to stop as soon as they compute the real solutions of the corresponding bivariate system. It was not easy to modify INSULATE and TOP to deal with general systems, so they were not executed on the first data set. TOP has a parameter that sets the initial precision (decimal digits). There is no easy way for choosing a good value. Hence, recorded its performance on initial values of 60 and 500 digits.

Experiments are not considered as competition, but as a crucial step for improving existing software. It is very difficult to compare different packages, since in most cases they are made for different needs. In addition, accurate timing in MAPLE is hard, since it is a general purpose package and a lot of overhead is added to its function calls. Lastly, the amount of experiments is not very large in order to draw safe conclusions.

Overall performance results are shown on table 1. In cases where the solvers failed to find the correct number of real solutions we indicate so with *. Note that in NEWMAC's column an additional step is required to distinguish the real solutions among the complex ones. In the sequel we refer only to G_RUR, since it is our faster implementation.

G_RUR is faster than FGB/RS in 8 out of the 18 instances, including C_5. The speedup factor ranges from 0.2 to 22 with an average of 2.62.

As for the three solvers from SYNAPS, G_RUR is faster than STURM in 6 out of the 18 instances, but it behaves worse usually in systems that are solved in < 100 msecs, because STURM is implemented in `C++`. As the dimension of the polynomial systems increases, G_RUR outperforms STURM and the latter's lack of fast algorithms for computing resultants becomes more evident. Overall, an average speedup of 2.2 is achieved. Compared with SUBDIV, G_RUR is faster in half of the instances and similarly to the previous case is slower on systems solved in < 400 msecs. On average, G_RUR achieves a speedup of 62.92 which is the result of the problematic behavior of SUBDIV in C_1 and W_1. If these

[2] `http://www-spaces.lip6.fr/index.html`

[3] `http://www-sop.inria.fr/galaad/logiciels/synaps/`

system	deg		solutions	Average Time (msecs)									
				BIVARIATE SOLVING							TOPOLOGY		
				SLV			FGB/RS	SYNAPS			INS	TOP	
	f	g		GRID	M_RUR	G_RUR		STURM	SUBDIV	NEWMAC		60	500
R_1	3	4	2	5	9	5	26	2	2	5	–	–	–
R_2	3	1	1	66	21	36	24	1	1	1	–	–	–
R_3	3	1	1	1	2	1	22	1	2	1	–	–	–
M_1	3	3	4	87	72	10	25	2	1	2	–	–	–
M_2	4	2	3	4	5	4	24	1	289∗	2	–	–	–
M_3	6	3	5	803	782	110	30	230	5,058∗	7	–	–	–
M_4	9	10	2	218	389	210	158	90	3∗	447	–	–	–
D_1	4	5	1	6	12	6	28	2	5	8	–	–	–
D_2	2	2	4	667	147	128	26	21	1∗	2	–	–	–
C_1	7	6	6	1,896	954	222	93	479	170,265∗	39	524	409	1,367
C_2	4	3	6	177	234	18	27	12	23∗	4	28	36	115
C_3	8	7	13	580	1,815	75	54	23	214∗	25	327	693	2,829
C_4	8	7	17	5,903	80,650	370	138	3,495	217∗	190∗	1,589	1,624	6,435
C_5	16	15	17	$> 20'$	60,832	3,877	4,044	$> 20'$	6,345∗	346∗	179,182	91,993	180,917
W_1	7	6	9	2,293	2,115	247	92	954	55,040∗	39	517	419	1,350
W_2	4	3	5	367	283	114	29	20	224∗	3	27	20	60
W_3	8	7	13	518	2,333	24	56	32	285∗	25	309	525	1,588
W_4	8	7	17	5,410	77,207	280	148	4,086	280∗	207∗	1,579	1,458	4,830

Table 1. Performance of our solvers and other tested software.

systems are omitted, then the speedup is 8.93 on average. NEWMAC is slower than G_RUR in M_4, D_1 and W_3 and comparable in R_1 and R_3. This time the average speedup of our implementation is 0.53. There are cases where NEWMAC may not compute some of the real solutions.

Finally, concerning the other MAPLE software, INSULATE is slower than G_RUR in all systems but W_2, thus our solver achieves an average speedup of 8.85. Compared to TOP with 60, resp. 500, digits, G_RUR is faster in all systems but W_2, yielding an average speedup of 7.79, resp. 22.64. Moreover, as the dimension of the polynomial systems increases, it becomes more efficient.

Acknowledgments. The authors thank the anonynous referees for their comments. All authors acknowledge partial support by IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-006413-2 (ACS - Algorithms for Complex Shapes). The third author is also partially supported by contract ANR-06-BLAN-0074 "Decotes".

References

[1] J. Abbott. Quadratic interval refinement for real roots. In *Proc. ACM Int. Symp. on Symbolic & Algebraic Comput.*, 2006. (Poster presentation), http://www.dima.unige.it/ ˜abbott/.

[2] D. Arnon and S. McCallum. A polynomial time algorithm for the topological type of a real algebraic curve. *J. Symbolic Computation*, 5:213–236, 1988.

[3] S. Basu, R. Pollack, and M-F.Roy. *Algorithms in Real Algebraic Geometry*, volume 10 of *Algorithms and Computation in Mathematics*. Springer-Verlag, 2nd edition, 2006.

[4] E. Brieskorn and H. Knörrer. *Plane Algebraic Curves*. Birkhäuser, Basel, 1986.

[5] P. Bürgisser, M. Clausen, and M.A. Shokrollahi. *Algebraic complexity theory*, volume 315 of *Grundlehren der Mathematischen Wissenschaften*. Springer-Verlag, Berlin, 1997.

[6] J. Canny. Some algebraic and geometric computations in PSPACE. In *Proc. ACM Symp. Theory of Computing*, pages 460–467, 1988.

[7] F. Cazals, J.-C. Faugère, M. Pouget, and F. Rouillier. The implicit structure of ridges of a smooth parametric surface. *Comput. Aided Geom. Des.*, 23(7):582–598, 2006.

[8] J. H. Davenport. Cylindrical algebraic decomposition. Technical Report 88–10, School of Mathematical Sciences, University of Bath, England, available at: http://www.bath.ac.uk/masjhd/, 1988.

[9] D. I. Diochnos, I. Z. Emiris, and E. P. Tsigaridas. On the complexity of real solving bivariate systems. In C. W. Brown, editor, *Proc. ACM Int. Symp. on Symbolic & Algebraic Comput.*, pages 127–134, Waterloo, Canada, 2007.

[10] D. I. Diochnos, I. Z. Emiris, and E. P. Tsigaridas. On the complexity of real solving bivariate systems. RR 6116, INRIA, February 2007. https://hal.inria.fr/inria-00129309.

[11] Z. Du, V. Sharma, and C. K. Yap. Amortized bound for root isolation via Sturm sequences. In D. Wang and L. Zhi, editors, *Int. Workshop on Symbolic Numeric Computing*, pages 113–129, School of Science, Beihang University, Beijing, China, 2005. Birkhauser.

[12] A. Eigenwillig, V. Sharma, and C. K. Yap. Almost tight recursion tree bounds for the Descartes method. In *Proc. ACM Int. Symp. on Symbolic & Algebraic Comput.*, pages 71–78, New York, NY, USA, 2006. ACM Press.

[13] A. Eigenwillig, M. Kerber, and N. Wolpert. Fast and exact geometric analysis of real algebraic plane curves. In C. W. Brown, editor, *Proc. ACM Int. Symp. on Symbolic & Algebraic Comput.*, pages 151–158, Waterloo, Canada, 2007.

[14] I. Z. Emiris and E. P. Tsigaridas. Real solving of bivariate polynomial systems. In V. Ganzha and E. Mayr, editors, *Proc. Computer Algebra in Scientific Computing (CASC)*, volume 3718 of *LNCS*, pages 150–161. Springer, 2005.

[15] I. Z. Emiris, B. Mourrain, and E. P. Tsigaridas. Real Algebraic Numbers: Complexity Analysis and Experimentation. In P. Hertling, C. Hoffmann, W. Luther, and N. Revol, editors, *Reliable Implementations of Real Number Algorithms: Theory and Practice*, volume 5045 of *LNCS*, pages 57–82. Springer Verlag, 2008. Also available in www.inria.fr/rrrt/rr-5897.html.

[16] L. González-Vega and M. El Kahoui. An improved upper complexity bound for the topology computation of a real algebraic plane curve. *J. Complexity*, 12(4):527–544, 1996.

[17] L. González-Vega and I. Necula. Efficient topology determination of implicitly defined algebraic plane curves. *Computer Aided Geometric Design*, 19(9):719–743, December 2002.

[18] L. González-Vega, H. Lombardi, T. Recio, and M-F. Roy. Sturm-Habicht Sequence. In *Proc. ACM Int. Symp. on Symbolic & Algebraic Comput.*, pages 136–146, 1989.

[19] K.H. Ko, T. Sakkalis, and N.M. Patrikalakis. Resolution of multiple roots of nonlinear polynomial systems. *International J. of Shape Modeling*, 11(1):121–147, 2005.

[20] T. Lickteig and M-F. Roy. Sylvester-Habicht Sequences and Fast Cauchy Index Computation. *J. Symb. Comput.*, 31(3):315–341, 2001.

[21] M. Mignotte and D. Ştefănescu. *Polynomials: An algorithmic approach*. Springer, 1999.

[22] P.S. Milne. On the solution of a set of polynomial equations. In B. Donald, D. Kapur, and J. Mundy, editors, *Symbolic and Numerical Computation for Artificial Intelligence*, pages 89–102. Academic Press, 1992.

[23] B. Mourrain and J-P. Pavone. Subdivision methods for solving polynomial equations. Technical Report RR-5658, INRIA Sophia-Antipolis, 2005. URL `www.inria.fr/rrrt/rr-5658.html`.

[24] B. Mourrain and Ph. Trébuchet. Solving projective complete intersection faster. In *Proc. ACM Int. Symp. on Symbolic & Algebraic Comput.*, pages 231–238. ACM Press, New York, 2000.

[25] B. Mourrain, S. Pion, S. Schmitt, J.-P. Técourt, E. Tsigaridas, and N. Wolpert. Algebraic issues in computational geometry. In J.-D. Boissonnat and M. Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, pages 117–155. Springer-Verlag, Mathematics and Visualization, 2006.

[26] V.Y. Pan. Univariate polynomials: Nearly optimal algorithms for numerical factorization and rootfinding. *J. Symbolic Computation*, 33(5):701–733, 2002.

[27] P. Pedersen, M-F. Roy, and A. Szpirglas. Counting real zeros in the multivariate case. In F. Eyssette and A. Galligo, editors, *Computational Algebraic Geometry*, volume 109 of *Progress in Mathematics*, pages 203–224. Birkhäuser, Boston, 1993. (Proc. MEGA '92, Nice).

[28] D. Reischert. Asymptotically fast computation of subresultants. In *ISSAC*, pages 233–240, 1997.

[29] J. Renegar. On the worst-case arithmetic complexity of approximating zeros of systems of polynomials. *SIAM J. Computing*, 18:350–370, 1989.

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

[31] T. Sakkalis. Signs of algebraic numbers. *Computers and Mathematics*, pages 131–134, 1989.

[32] T. Sakkalis and R. Farouki. Singular points of algebraic curves. *J. Symb. Comput.*, 9(4):405–421, 1990.

[33] M. van Hoeij and M. Monagan. A modular GCD algorithm over number fields presented with multiple extensions. In *ISSAC*, pages 109–116, July 2002.

[34] J. von zur Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge Univ. Press, Cambridge, U.K., 2nd edition, 2003.

[35] J. von zur Gathen and T. Lücking. Subresultants revisited. *Theor. Comput. Sci.*, 1-3(297):199–239, 2003.

[36] N. Wolpert. *An Exact and Efficient Approach for Computing a Cell in an Arrangement of Quadrics*. PhD thesis, MPI fuer Informatik, October 2002.
[37] N. Wolpert and R. Seidel. On the Exact Computation of the Topology of Real Algebraic Curves. In *Symposium of Computational Geometry*. ACM, 2005. to appear.
[38] C.K. Yap. *Fundamental Problems of Algorithmic Algebra*. Oxford University Press, New York, 2000.