HAL open science

On the complexity of real solving bivariate systems

Dimitrios Diochnos, Ioannis Z. Emiris, Elias Tsigaridas

To cite this version:

Dimitrios Diochnos, Ioannis Z. Emiris, Elias Tsigaridas. On the complexity of real solving bivariate systems. [Research Report] RR-6116, INRIA. 2007. inria-00129309v5

HAL Id: inria-00129309

https://inria.hal.science/inria-00129309v5

Submitted on 16 Oct 2007

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

On the complexity of real solving bivariate systems

Dimitrios I. Diochnos — Ioannis Z. Emiris — Elias P. Tsigaridas

N° 6116

June 2007

Thème SYM

Rapport de recherche

ISSN 0249-6399

On the complexity of real solving bivariate systems

Dimitrios I. Diochnos* , Ioannis Z. Emiris† , Elias P. Tsigaridas‡ §

Thème SYM — Systèmes symboliques
Projet VEGAS

Rapport de recherche n° 6116 — June 2007 — 55 pages

Abstract: This paper is concerned with exact real solving of well-constrained, bivariate algebraic 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: we ignore polylogarithmic factors, and N bounds the degree and the bitsize of the polynomials. The previous record bound was $\widetilde{\mathcal{O}}_B(N^{16})$.

Our main tool is signed subresultant sequences, extended to several variables by the technique of binary segmentation. We exploit recent advances on the complexity of univariate root isolation, and extend them to multipoint evaluation, to sign evaluation of bivariate polynomials over two 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 $\widetilde{\mathcal{O}}_B(N^{16})$.

All algorithms have been implemented in MAPLE, in conjunction with numeric filtering. We compare them against FGB/RS and system solvers from SYNAPS; we also consider 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: MAPLE code, real solving, polynomial systems, real algebraic numbers, complexity

This work is partially supported by the european project ACS (Algorithms for Complex Shapes, IST FET Open 006413) and ARC ARCADIA (http://www.loria.fr/∼petitjea/Arcadia/)

* Department of Informatics and Telecommunications National Kapodistrian University of Athens, HELLAS
† Department of Informatics and Telecommunications National Kapodistrian University of Athens, HELLAS
‡ LORIA-INRIA Lorraine, Nancy, FRANCE
§ The work on this project started while at INRIA Sophia-Antipolis.

On the complexity of real solving bivariate systems

Résumé : On s'intéresse à la résolution exacte, dans les réels, de systèmes algébriques bien-contraints en deux variables. Le problème principal est d'isoler les racines communes en des rectangles rationaux, et de calculer leur multiplicités. Nous présentons trois algorithmes et nous analysons leur complexité asymptotique binaire, arrivant à une borne de $\widetilde{\mathcal{O}}_B(N^{14})$ pour la méthode de projection, et de $\widetilde{\mathcal{O}}_B(N^{12})$ pour les deux méthodes basées sur le résultant: ces bornes ignorent les facteurs poly-logarithmiques, où N borne le degré et la taille binaire des polynômes. La borne précedente était de $\widetilde{\mathcal{O}}_B(N^{16})$.

Notre outil principal sont les séquences de sous-résultants, généralisées en plusieurs variables. Nous nous basons sur les avancés récentes sur la complexité d'isolation univariée, et nous généralisons ces résultats pour borner la complexité de l' évaluation en plusieurs points, pour déterminer le signe de polynômes en deux variables, et pour compter le nombre de racines d'un polynôme en coefficients algébriques. Nos algorithmes s'appliquent au problème d'inégalités simultanées; ils calculent aussi la topologie d'une courbe algébrique dans le plan en $\widetilde{\mathcal{O}}_B(N^{12})$, tandis que la borne existante était de $\widetilde{\mathcal{O}}_B(N^{16})$.

Nos algorithmes ont tous été implementés en Maple, avec filtrage numérique. Nous les comparons avec GBRS, des solveurs de SYNAPS, ainsi que les programmes INSULATE et TOP en MAPLE, qui calculent la topologie d'une courbe. Notre logiciel est parmi les plus robustes et ses temps de calculs sont comparables, ou plus lents d'un petit facteur constant, par rapport aux logiciels C/C++.

Mots-clés : code Maple, résolution dans les réels, systèmes algébriques, nombres algébriques, complexité

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 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, $\mathcal{O}_B$ 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 $\widetilde{\mathcal{O}}_B(N^{14})$ [GVEK96], see also [BPM06], 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 [GVEK96] 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 $\widetilde{\mathcal{O}}_B(N^6)$ [DSY05, ESY06, EMT07]. This brought complexity closer to $\widetilde{\mathcal{O}}_B(N^4)$, which is achieved by numerical methods [Pan02].

In [KSP05], 2×2 systems are solved and the multiplicities computed under the assumption that a generic shear has been obtained, based on [SF90]. In [Wol02], 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 [ET05], except that matching and multiplicity computation was simpler. In [MP05], 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. [Mil92, PRS93].

Determining the topology of a real algebraic plane curve is a closely related problem. The best bound is $\widetilde{\mathcal{O}}_B(N^{14})$ [BPM06, GVEK96]. In [WS05] three projections are used; this is implemented in INSULATE, with which we make several comparisons. Work in [Ker06] offers an efficient implementation of resultant-based methods. For an alternative using Gröbner bases see [CFPR06]. To the best of our knowledge the only result in topology determination using isolating intervals is [AM88], 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 [GVEK96] 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, note that the input is in $\mathcal{O}_B(N^3)$, and the bitsize of all output isolation points for univariate solving is $\widetilde{\mathcal{O}}_B(N^2)$, 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 (lem. 2.7). We establish a complexity bound for bivariate sign evaluation (th. 3.7), which helps us derive bounds for root counting in an extension field (lem. 5.1) and for the problem of simultaneous inequalities (cor. 5.4). We study the complexity of bivariate polynomial real solving, using three projection-based algorithms: a straightforward grid method (th. 4.1), a specialized RUR approach (th. 4.6), and an improvement of the latter using fast GCD (th. 4.7). 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. 5.6).

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 seem very 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 robust and effective; 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.

2 Univariate polynomials

For $f \in \mathbb{Z}[y_1, \ldots, y_k, x]$, $\mathsf{dg}(f)$ denotes its total degree, while $\deg_x(f)$ 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, $\mathsf{M}(\tau) = \widetilde{\mathcal{O}}_B(\tau)$, $\mathsf{M}(d, \tau) = \widetilde{\mathcal{O}}_B(d\tau)$.

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 $(Q_0, \ldots, Q_{k-1}, R_k)$.

Here, we consider signed subresultant sequences, which contain polynomials similar to the polynomials in the signed polynomial remainder sequence; see [vzGL03] 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. [GVLRR89]. 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, and by $\mathbf{SR}(f, g; \mathsf{a})$ 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 2.1 *[LR01, LRSED00, Rei97] Assuming $p \geq q$, $\mathbf{SR}(f, g)$ is computed in $\widetilde{\mathcal{O}}_B(p^2 q\tau)$ and $\mathcal{L}\,(\mathbf{SR}_j(f, g)) = \mathcal{O}(p\tau)$. For any f, g, their quotient boot, any polynomial in $\mathbf{SR}(f, g)$, their resultant, and their* gcd *are computed in $\widetilde{\mathcal{O}}_B(pq\tau)$.*

Proposition 2.2 *[LR01, Rei97] Let $p \geq q$. We can compute $\mathbf{SR}(f, g; \mathsf{a})$, where $\mathsf{a} \in \mathbb{Q} \cup \{\pm\infty\}$ and $\mathcal{L}\,(\mathsf{a}) = \sigma$, 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 $\mathbf{SR}$ is the same as that of computing $\mathbf{SRQ}(f, g)$ (Pr. 2.1).

We follow Lickteig and Roy [LR01]. For two polynomials A, B of degree bounded by D and bit size bounded by L, we can compute $\mathbf{SR}(A, B)(\mathsf{a})$, where $\mathcal{L}\,(\mathsf{a}) \leq L$, in $\widetilde{\mathcal{O}}_B(\mathsf{M}\,(D, L))$. In our case $D = \mathcal{O}(q)$ and $L = \mathcal{O}(p\tau + q\sigma)$, thus the total costs 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. 2.2 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. [BPM06, GVLRR89]) sign variations.

Corollary 2.3 *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.

Proposition 2.4 *[DSY05, ESY06, EMT07] 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 $\mathcal{L}(f_{red}) = \mathcal{O}(p + \tau_f)$.*

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

Corollary 2.5 *[BPM06, EMT07] 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 $\mathsf{dg}(g) = q, \mathcal{L}(g) = \tau_g$, we compute $\operatorname{sign}(g(\alpha))$ in bit complexity $\widetilde{\mathcal{O}}_B(pq\max\{\tau_f, \tau_g\} + p\min\{p, q\}^2\tau_f)$.*

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

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

Actually the previous relation holds in a more general context, when f *dominates* g, see [Yap00] for details. Notice that $\operatorname{sign}(f'(\alpha)) = \operatorname{sign}(f(\mathsf{b})) - \operatorname{sign}(f(\mathsf{b}))$, 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 biggest degree, i.e the first (and the second if $p < q$) of $\mathbf{SR}(f, g)$ over a and b. Thus the complexity is that of cor. 2.3, viz.

$$\widetilde{\mathcal{O}}_B(pq\max\{\tau_f, \tau_g\} + \min\{p, q\}^2 p\,\tau_f)$$

Thus the complexity of the operation is 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 $\operatorname{sign}(g(\alpha)) = 0$. Otherwise we proceed as above. □

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. [Yap00]. The following lemma, derived from Davenport-Mahler's bound, is crucial.

Lemma 2.6 (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.[of lem. 2.6] Let there be $r \leq p$ real roots. The isolating point between two consecutive real roots α_j, 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 $\frac{1}{2^r}\prod_j \Delta_j$. Using the Davenport-Mahler bound, $\prod_j \Delta_j \geq 2^{-\mathcal{O}(p^2+p\tau_f)}$ and we take logarithms. □

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. [EMT07], is to apply cor. 2.5 r times, where r is the number of real roots of f. But we can do better by applying lem. 2.6:

Lemma 2.7 *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.[of lem. 2.7] Let s_j be the bitsize of the j-th endpoint, where $0 \le j \le r$. The evaluation of $\mathbf{SR}(f,g)$ over this endpoint, by cor. 2.3, costs $\widetilde{\mathcal{O}}_B(pq\tau + \min\{p,q\}^2 s_j)$. To compute the overall cost, we should sum over all the isolating points. The first summand is $\widetilde{\mathcal{O}}_B(p^2 q\tau)$. By pr. 2.6, the second summand becomes $\widetilde{\mathcal{O}}_B(\min\{p,q\}^2(p^2 + p\tau_f))$ and is dominated. □

3 Multivariate polynomials

We discuss multivariate polynomials, using binary segmentation [Rei97]. An alternative approach could be [Klo95]. Let $f, g \in (\mathbb{Z}[y_1, \ldots, y_k])[x]$ with $\mathtt{dg}_x(f) = p \ge q = \mathtt{dg}_x(g)$, $\mathtt{dg}_{y_i}(f) \le d_i$ and $\mathtt{dg}_{y_i}(g) \le d_i$. Let $d = \prod_{i=1}^k d_i$ and $\mathcal{L}(f), \mathcal{L}(g) \le \tau$. The y_i-degree of every polynomial in $\mathbf{SR}(f,g)$, is bounded by $\mathtt{dg}_{y_i}(\mathtt{res}(f,g)) \le (p+q)d_i$. 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 $\mathbf{SR}(f,g)$. Now $\psi(f), \psi(g) \in (\mathbb{Z}[y])[x]$ have $y-$degree $\le (p+q)^{k-1} d$ since, in the worst case, f or g hold a monomial such as $y_1^{d_1} y_2^{d_2} \ldots y_k^{d_k}$. Thus, $\mathtt{dg}_y(\mathtt{res}(\psi(f), \psi(g))) < (p+q)^k d$.

Proposition 3.1 *[Rei97] We can compute $\mathbf{SRQ}(f,g)$, any polynomial in $\mathbf{SR}(f,g)$, and $\mathtt{res}(f,g)$ in $\widetilde{\mathcal{O}}_B(q(p+q)^{k+1} d\tau)$.*

Lemma 3.2 *$\mathbf{SR}(f,g)$ is computed 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 $\chi : \mathbb{Z}[y] \mapsto \mathbb{Z}$, such that $y \mapsto 2^{\lceil c(p+q)\tau \rceil}$, and let $\phi = \psi \circ \chi : \mathbb{Z}[y_1, y_2 \ldots, y_k] \to \mathbb{Z}$. Then $\mathcal{L}(\phi(f)), \mathcal{L}(\phi(g)) \le \mathsf{c}\,(p+q)^k\, d\,\tau$. Now apply pr. 2.1. □

Theorem 3.3 *We can evaluate $\mathbf{SR}(f,g)$ at $x = \alpha$, where $\mathsf{a} \in \mathbb{Q} \cup \{\infty\}$ and $\mathcal{L}(a) = \sigma$, in $\widetilde{\mathcal{O}}_B(q(p+q)^{k+1} d \max\{\tau, \sigma\})$.*

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: $\mathrm{sign}(F(\alpha, \beta))$
compute $\mathbf{SRQ}_x(A, F)$
$L_1 \leftarrow \mathbf{SR}_x(A, F\,;\, \mathsf{a}_1)$, $V_1 \leftarrow \emptyset$
foreach $f \in L_1$ **do** $V_1 \leftarrow$ ADD$(V_1,$ SIGN_AT$(f, \beta))$
$L_2 \leftarrow SR_x(A, F\,;\, \mathsf{a}_2)$, $V_2 \leftarrow \emptyset$
foreach $f \in L_2$ **do** $V_2 \leftarrow$ ADD$(V_2,$ SIGN_AT$(f, \beta))$
RETURN $(\mathrm{VAR}(V_1) - \mathrm{VAR}(V_2)) \cdot \mathrm{sign}(A'(\alpha))$

Proof. Compute $\mathbf{SRQ}(f, g)$ in $\widetilde{\mathcal{O}}_B(q(p+q)^{k+1} d\,\tau)$ (prop. 3.1), then evaluate it over a, using binary segmentation. For this 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 $x = \mathsf{a}$ yields values of size $\widetilde{\mathcal{O}}(p\sigma)$. After the evaluation we obtain polynomials in $\mathbb{Z}[y_1, \ldots, y_k]$ with coefficient bitsize bounded by $\max\{(p + q)\tau, p\sigma\} \leq (p + q) \max\{\tau, \sigma\}$.

Consider $\chi : \mathbb{Z}[y] \to \mathbb{Z}$, such that $y \mapsto 2^{\lceil \mathsf{c}(p+q) \max\{\tau,\sigma\} \rceil}$, for a suitable constant c. Apply the map $\phi = \psi \circ \chi$ to f, g. Now, $\mathcal{L}(\phi(f)), \mathcal{L}(\phi(g)) \leq \mathsf{c}d(p + q)^k \max\{\tau, \sigma\}$. By prop. 2.2, the evaluation costs $\widetilde{\mathcal{O}}_B(q(p + q)^{k+1} d \max\{\tau, \sigma\})$. □

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

Corollary 3.4 *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* $\mathbf{SR}(f, g)$, $\mathsf{dg}_x(\mathbf{SR}_j(f, g)) = \mathcal{O}(\max\{p, q\})$, $\mathsf{dg}_y(\mathbf{SR}_j(f, g)) = \mathcal{O}(\max\{p, q\}d)$, *and also* $\mathcal{L}(\mathbf{SR}_j(f, g)) = \mathcal{O}(\max\{p, q\}\tau)$.

Corollary 3.5 *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 3.6 *We 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, *we have* $\mathsf{dg}_y(\mathbf{SR}_j(f, g\,;\, \mathsf{a})) = \mathcal{O}((p + q)d)$ *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 [Sak89], and generalizes the univariate case, e.g. [EMT07, Yap00]. 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 3.7 (sign_at) *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 $\mathbf{SRQ}_x(A, F)$ so as to evaluate $\mathbf{SR}(A, F)$ on the endpoints of α, in $\widetilde{\mathcal{O}}_B(n_1^2 n_2^2 \sigma)$ (cor. 3.5).

We compute $\mathbf{SR}(A, F; \mathsf{a}_1)$. The first polynomial in the sequence is A, but we already know its value on a_1. This computation costs $\widetilde{\mathcal{O}}_B(n_1^2\, n_2^3\, \sigma)$ by cor. 3.6 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. 2.5, 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. 2.1), $\mathsf{dg}(g) = \mathcal{O}(n_2)$ 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. 2.5, 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 $\mathrm{sign}(A'(\alpha))$, which is known from the process that isolated α. If the last sign in the two sequences is alternating, then $\mathrm{sign}(F(\alpha, \beta)) = 0$. □

4 Bivariate real solving

Let $F, G \in \mathbb{Z}[x, y]$, $\mathsf{dg}(F) = \mathsf{dg}(G) = n$ 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 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.

4.1 The GRID algorithm

Algorithm GRID is straightforward, see also [ET05, Wol02]. We compute the $x-$ and $y-$coordinates of the real solutions, as real roots of the resultants $\mathtt{res}_x(F, G)$ and $\mathtt{res}_y(F, G)$. Then, we match them using the algorithm SIGN_AT (th. 3.7) 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-aligned 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 SIGN_AT (alg. 1) is not efficient. However, its simplicity makes it attractive. The algorithm requires

Algorithm 2: GRID(F, G)

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

$R_x \leftarrow \mathtt{res}_y(F, G)$
$L_x, M_x \leftarrow$ SOLVE(R_x)
$R_y \leftarrow \mathtt{res}_x(F, G)$
$L_y, M_y \leftarrow$ SOLVE(R_y)
$Q \leftarrow \emptyset$
foreach $\alpha \in L_x$ **do**
  foreach $\beta \in L_y$ **do**
    if SIGN_AT$(F, \alpha, \beta) = 0 \wedge$ SIGN_AT$(G, \alpha, \beta) = 0$ **then** $Q \leftarrow$ ADD$(Q, \{\alpha, \beta\})$
RETURN Q

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 α by lem. 5.1.

Theorem 4.1 *Isolating all real roots of system $F = G = 0$ using* GRID *has complexity $\widetilde{\mathcal{O}}_B(n^{14} + n^{13}\sigma)$, provided $\sigma = O(n^3)$.*

Proof. First we compute the resultant of F and G w.r.t. y, i.e. R_x. The complexity is $\widetilde{\mathcal{O}}_B(n^4\sigma)$, using cor. 3.5. Notice that $\mathtt{dg}(R_x) = \mathcal{O}(n^2)$ and $\mathcal{L}(R_x) = \mathcal{O}(n\,\sigma)$. We isolate its real roots in $\widetilde{\mathcal{O}}_B(n^{12} + n^{10}\sigma^2)$ (prop. 2.4) and store them in L_x. 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 $\mathcal{O}(n^2 + n\,\sigma)$ [BPM06, EMT07]. The isolating intervals have endpoints of size $\mathcal{O}(n^4 + n^3\,\sigma)$.

Let r_x, resp. r_y be the number of real roots of the corresponding resultants. Both are bounded by $\mathcal{O}(n^2)$. 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 SIGN_AT (th. 3.7). Each evaluation 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 [BK86, 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.

The algorithm reduces to bivariate sign determination and does not require bivariate factorization. We shall use the resultant, since it allows for multiplicities to "project". Previous work includes [GVEK96, SF90, WS05]. The sum of multiplicities of all roots (α, β_j)

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.

4.1.1 Deterministic shear.

We determine an adequate (horizontal) shear such that

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

when $t \mapsto t_0 \in \mathbb{Z}$, has simple roots corresponding to the projections of the common roots of the system $F(x, y) = G(x, y) = 0$ 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 $\Delta(t_0) \neq 0$.

Example 4.2 *Take the circle $f = (x-1)^2 + y^2 - 1$, and the double line $g = y^2$ with two double roots $(0, 0), (2, 0)$. We shall project roots on the y-axis under the vertical shear:*

$$f(x, y+tx) = x^2(1+t^2) + 2x(ty-1) + y^2, g(x, y+tx) = y^2 + 2txy + (tx)^2.$$

Then, $R(y) = y^2[y^2(t^4+1) + 2t^3y + t^2]$. The square-free part is $R_{red}(y) = y[y^2(t^4+1) + 2t^3y + t^2]$ and $\Delta(t) = t^6(t^4+1)(3t^4+4)$. Clearly, one must avoid the value $t = 0$, but any other integer is valid.

Lemma 4.3 *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 we compute the resultant w.r.t. y in $\widetilde{\mathcal{O}}_B(n^5\sigma)$, which is a polynomial in $\mathbb{Z}[t, x]$, of degree $\mathcal{O}(n^2)$ and bitsize $\widetilde{\mathcal{O}}(d\sigma)$. We consider this polynomial as univariate in x and we compute first 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 $\widetilde{\mathcal{O}}(d^4 + d^3\sigma)$.

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. 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 [DET07]. 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 [GVN02], also [BPM06]. 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 4.4 *Consider the setting of th. 4.1. Having isolated all real roots of $F = G = 0$, it is possible to determine their multiplicities in $\widetilde{\mathcal{O}}_B(n^{12} + n^{11}\sigma + n^{10}\sigma^2)$.*

Proof.[of th. 4.4] 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 $\widetilde{\mathcal{O}}_B(n^{12}+n^{10}\sigma^2)$. 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 pr. 2.6, $\sum_i s_i = \mathcal{O}(n^4 + n^3\sigma)$.

Our analysis resembles that of [EMT07] for proving pr. 2.4. 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 a 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 4.5 *[GVEK96, BPM06] 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 [Ren89, Can88, Rou99, BPM06]; it generalizes the primitive element method by Kronecker. Here we adapt it to small-dimensional systems.

Our algorithm is similar to [GVN02, GVEK96]. However, their algorithm computes only a RUR using prop. 4.5, 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 [ET05], so that the output includes isolating rectangles, hence the name modified-RUR (M_RUR). The most important

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

difference with [GVEK96] is that they represent algebraic numbers by Thom's encoding while we use isolating intervals, which were thought of having high theoretical complexity. We will prove that this is not the case.

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. 4.5. First we compute the sequence $\mathbf{SR}(F, G)$ w.r.t. y in $\widetilde{\mathcal{O}}_B(n^5\,\sigma)$ (cor. 3.4).

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 $\mathcal{O}(n^3\,\sigma)$:

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

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. 4.5, for every α_i we must compute $k \in \mathbb{N}^*$ such the assumptions of the theorem are fulfilled; this is possible by genericity. We follow [MPS+06, GVEK96] 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 [MS99], $\mathcal{L}(\Gamma_j) = \mathcal{O}(n^2 + n\sigma)$. 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. 2.1) 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. 2.6) 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 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 üôé $\beta_j = A(\alpha)$, 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. 2.5). 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 4.6 *We isolate all real roots of $F = G = 0$, if F, G are in generic position, by* M_RUR *in $\widetilde{\mathcal{O}}_B(n^{12} + n^{10}\sigma)$.*

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 previous section. The bitsize of polynomials of the (sheared) system becomes $\widetilde{\mathcal{O}}(n + \sigma)$ [GVEK96] and does not change the bound of th. 4.6. However, now is raised the problem of expressing the real roots in the original coordinate system (see also the proof of th. 4.4).

4.3 The G_RUR algorithm

We present an algorithm that uses some ideas from RUR but relies on GCD computations of polynomials with coefficients in an extension field to achieve efficiency (hence the name G_RUR). For the GCD computations we use the algorithm (and the implementation) of [vHM02].

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 [vHM02] 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. The complexity 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 have to do 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 $\widetilde{\mathcal{O}}_B(n + \sigma)$ [BPM06].

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 $m = \mathcal{O}(n^2 + n\sigma)$ and 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 [MS99]. Notice that H is a square-free polynomial in $(\mathbb{Z}[\alpha])[y]$, of degree $\mathcal{O}(n)$ and bitsize $\mathcal{O}(n^2 + n\sigma)$, the real roots of which correspond to the real solutions of the system with abscissa α. 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 $\mathcal{O}(n)$ 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. 2.5 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. 2.6, 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 4.7 *We can isolate the real roots of the system $F = G = 0$, using* G_RUR *in $\widetilde{\mathcal{O}}_B(n^{12} + n^{10}\sigma)$.*

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 $\alpha = (A, [\mathsf{a}_1, \mathsf{a}_2])$ and $\beta = (B, [\mathsf{b}_1, \mathsf{b}_2])$, 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 $n_1^2 = \mathcal{O}(n_2)$. 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 [LR01, GVLRR89, GVEK96].

Using Sturm's theorem, e.g. [BPM06, Yap00], 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 $\mathbb{Z}(\alpha)$.

The above procedure is equivalent, due to the good specialization properties of subresultants [BPM06, GVLRR89], to that of 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. 3.5 with $p = q = d = n_1$, and $\tau = \sigma$. The sequence $\mathbf{sr}$, 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. 2.5 with $p = n_2$, $q = n_1^2$, $\tau_f = \tau$, and $\tau_g = n_1\sigma$. This proves the following:

Lemma 5.1 *We count the number of real roots of $\bar{F}$ in $\widetilde{\mathcal{O}}_B(n_1^4 n_2\sigma + n_1^5 n_2\tau)$.*

In order to compute the number of 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. 3.4 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 can compute the sign of each of them evaluated it over (α, β) in $\widetilde{\mathcal{O}}_B(n_1^4 n_2^3 \max\{n_1\sigma, \tau\})$ (th. 3.7). This proves the following:

Lemma 5.2 *We can 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 the signs of these polynomials evaluated 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. 3.6 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. 2.5 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 5.3 *We can 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 $1 \le i \le \ell_1, 1 \le j \le \ell_2, 1 \le k \le \ell_3$. Let $\ell = \ell_1 + \ell_2 + \ell_3$.

Corollary 5.4 *There is an algorithm that solves the problem of ℓ simultaneous inequalities of degree $\le n$ and bitsize $\le \sigma$, in $\widetilde{\mathcal{O}}_B(\ell n^{12} + \ell n^{11}\sigma + n^{10}\sigma^2)$.*

Proof.[of cor. 5.4] 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 $\mathrm{sign}(A_i(\alpha, \beta))$, $\mathrm{sign}\,(B_i(\alpha, \beta))$ and $\mathrm{sign}\,(C_i(\alpha, \beta))$. Each sign evaluation costs $\widetilde{\mathcal{O}}_B(n^{10}+n^9\sigma)$, using th. 3.7 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.

We improve the complexity of computing the topology of a real plane algebraic curve. See [BPM06, GVEK96, MPS+06] for the algorithm.

In studying Algebraic curves we use the following:

Lemma 5.5 *Given $f \in \mathbb{Z}[x, y]$, it holds $\frac{d}{dx} f(x, y)\big|_{x=x+ty} = \frac{d}{dx} f(x + ty, y), t \in \mathbb{Z}$.*

Proof. It holds that:

$$\frac{d}{dx} f(x, y) = \frac{d}{dx} \sum_{i=0}^{N} a_i x^i h_i(y) = \sum_{i=1}^{N} a_i h_i(y) i x^{i-1}.$$

and

$$\begin{aligned} \frac{d}{dx} f(x + ty, y) &= \frac{d}{dx} \sum_{i=1}^{N} a_i (x + ty)^i h_i(y) + \frac{d}{dx} h_0(y) \\ &= \sum_{i=1}^{N} a_i h_i(y) i (x + ty)^{i-1}. \end{aligned}$$

□

However, assuming that f depends on x the shear transform does not commute with differentiation wrt y i.e. $\frac{d}{dy} f(x, y)\Big|_{x=x+ty} \neq \frac{d}{dy} f(x + ty, y)$. For a counter-example, take $f = x$.

We consider the curve, in generic position, 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 $\widetilde{\mathcal{O}}_B(n^{12} + n^{10}\sigma^2)$. Next, we compute the intermediate points on the x axis, in $\widetilde{\mathcal{O}}_B(n^4 + n^3\sigma)$ (lem. 2.6). 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. 2.2 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. 5.1, 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 $y = \beta$. We do this by lem. 5.2, in $\widetilde{\mathcal{O}}_B(n^{13} + n^{12}\sigma)$. 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 [BPM06].

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 [GVEK96]. Using lem. 2.7 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 5.6 *We compute the topology of a real plane algebraic curve, defined by a polynomial of degree n and bitsize σ, in $\widetilde{\mathcal{O}}_B(n^{12} + n^{11}\sigma + n^{10}\sigma)$.*

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 the open source MAPLE implementation[1] that was created as part of this thesis and illustrates its capabilities through comparative experiments. The design is based on object oriented programming and the generic programming paradigm in view of transferring the implementation to `C++` in the future.

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

The class of real algebraic numbers represents them in isolating interval representation. We provide algorithms for computing *signed* polynomial remainder sequences; more particularly euclidean, primitive-part, subsresultant and Sturm-Habicht sequences. In addition to that, we perform real solving of univariate polynomials using Sturm's algorithm, and allow computations with one and two real algebraic numbers, such as sign evaluation and comparison. Finally, the current implementation exhibits the algorithms for real solving of bivariate systems that were mentioned in section 4.

However, in order to speedup the various computations and create a more real-world library, filtering techniques have been used. For this purpose, two instances of the rational endpoints that define the isolating intervals of the various real algebraic numbers are stored; one pair of endpoints (usually with larger bitsize) is used in filtering techniques, while the other one is used for exact computations via Sturm sequences.

6.1 Augmenting performance

This section is devoted to the filtering techniques that are currently used in the library.

A. **Pre-computation filtering in** M_RUR
Recall that M_RUR binary-searches for solutions along the y-axis. For this reason the intervals of candidate solutions along the x-axis are refined [Abb06] in order to help the interval arithmetic filters (refer to the following paragraph) that will be used inside the FIND procedure.

B. **Interval Arithmetic**
In cases where one wants to compute the sign of a polynomial evaluated at a real algebraic number, the first attempt is to yield the result via interval arithmetic techniques. The reader may refer to [Neu90] for details in the evaluations that arise. This filter is applied heuristically several times, based on the total degree of the input polynomials, with a combination of quadratic refinement of the defining intervals [Abb06] between executions in each loop.

C. **GCD**
In cases where the above filter fails to yield a result and one either wants to compare two real algebraic numbers or perform univariate SIGN_AT the gcd of the two polynomials that are involved is computed. By definition, the gcd of the two polynomials has a root in (the intersection of) the intervals if and only if both polynomials have a same root, in which case the two numbers are equal, or equivalently the required sign is zero.

Concluding, if both of the above filtering techniques fail, the library switches to exact and costly computations via Sturm sequences. Note however, that in these computations the rational endpoints with higher bitsize that have arisen through the above filtering techniques are not used; instead the initial endpoints with smaller bitsize are used.

6.2 Bivariate solving and SLV library

In order to evaluate the implementation we have performed tests with the polynomial systems that are presented in section A.1. The performance of the implemented algorithms for bivariate solving is averaged over 10 iterations in MAPLE 9.5 console and is shown in table 1. Polynomial systems R_i, M_i, and D_i are presented in [ET05], systems C_i in [GVN02], and W_i are the C_i after swapping the x and y variables. Note that systems C_i and W_i are of the form $f = \frac{\partial f}{\partial y} = 0$ that arise in the topology of real plane algebraic curves. Finally, the polynomial system W_5 is not generated since the initial curve is a symmetric polynomial.

system	**deg**		$\mathbb{R}_{\mathbf{alg}}$	**Average Time (msecs)**		
	f	**g**	**solutions**	GRID	M_RUR	G_RUR
R_1	3	4	2	5	9	5
R_2	3	1	1	66	21	36
R_3	3	1	1	1	2	1
M_1	3	3	4	87	72	10
M_2	4	2	3	4	5	4
M_3	6	3	5	803	782	110
M_4	9	10	2	218	389	210
D_1	4	5	1	6	12	6
D_2	2	2	4	667	147	128
C_1	7	6	6	$1,896$	954	222
C_2	4	3	6	177	234	18
C_3	8	7	13	580	$1,815$	75
C_4	8	7	17	$5,903$	$80,650$	370
C_5	16	15	17	$> 20'$	$60,832$	$3,877$
W_1	7	6	9	$2,293$	$2,115$	247
W_2	4	3	5	367	283	114
W_3	8	7	13	518	$2,333$	24
W_4	8	7	17	$5,410$	$77,207$	280

Table 1: Performance averages over 10 runs in MAPLE 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

Recall that computations are performed first using intervals with floating point arithmetic (as it was described in section 6.1) and, if they fail, then an exact algorithm using rational arithmetic is called. For GCD computations in an extension field the MAPLE package of [vHM02] is used. Finally, also note that the optimal algorithms for computing and evaluating polynomial remainder sequences have not yet been implemented. Hence, it is reasonable to expect more efficient computations on a future release of the library.

It seems that 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 less than a second, apart from system C_5. Overall, for total degrees ≤ 8, G_RUR requires less than 0.4 secs to respond. On average, G_RUR is 7-11 times faster than GRID, and about 38 times than M_RUR. The inefficiency of M_RUR can be justified by the fact that M_RUR solves sheared systems which are dense and of increased bitsize w.r.t. the original systems. Finally, it should be noted that GRID reaches a stack limit with the default MAPLE stack size $(8, 192)$ when trying to solve system C_5. However, even when we increased the stack ten times, GRID could not yield all solutions within 20 minutes. Setting the stack size to the required limit can be done with the following MAPLE command:

```
kernelopts(stacklimit=81920);
```

6.2.1 Comparing SLV solvers

The following two paragraphs will briefly compare G_RUR with GRID and M_RUR in bivariate solving.

G_RUR VS. GRID Table 2 presents running times for bivariate solving between GRID and G_RUR. The final column in this table indicates the speedup that is achieved when preferring G_RUR for bivariate solving. In other words, speedup $= \frac{\text{TIME}_{\text{GRID}}}{\text{TIME}_{\text{G_RUR}}}$. As it is shown from the table G_RUR can be up to 21.58 times faster than GRID with an average speedup of around 7.27 among the input systems and excluding system C_5 where GRID failed to reply within 20 minutes. Moreover, in terms of total computing times for the entire test-set (again excluding system C_5) we can observe that:

- Total time for GRID $= 19,001$ msecs.
- Total time for G_RUR $= 1,860$ msecs.

In other words, the speedup in terms of total computing time is about 10.22.

G_RUR VS. M_RUR Table 3 presents running times for bivariate solving between M_RUR and G_RUR. Similarly with the previous table, the final column indicates the speedup that is achieved when preferring G_RUR for bivariate solving. As it is shown from the table G_RUR can be up to 275.74 times faster than M_RUR with an average speedup of around 38.01 among the input polynomial systems. Moreover, in terms of total computing times for the entire test-set we can observe that:

- Total time for M_RUR $= 227,862$ msecs.
- Total time for G_RUR $= 5,737$ msecs.

system	Average Time		speedup
	GRID	G_RUR	
R_1	5	5	1.00
R_2	66	36	1.83
R_3	1	1	1.00
M_1	87	10	8.70
M_2	4	4	1.00
M_3	803	110	7.30
M_4	218	210	1.04
D_1	6	6	1.00
D_2	667	128	5.21
C_1	$1,896$	222	8.54
C_2	177	18	9.83
C_3	580	75	7.73
C_4	$5,903$	370	15.95
C_5	$> 20'$	$3,877$	$-$
W_1	$2,293$	247	9.28
W_2	367	114	3.22
W_3	518	24	21.58
W_4	$5,410$	280	19.32

Table 2: The performance of GRID and G_RUR implementations on bivariate solving and the speedup that is achieved when choosing G_RUR.

In other words, the speedup in terms of total computing time is about 39.72.

Again, it should be noted that M_RUR solves sheared systems which are dense and of increased bitsize. In addition to that, since the polynomial systems are sheared (whenever necessary) in M_RUR's case, M_RUR also computes the multiplicities on the intersections. A more accurate comparison will follow when all solvers will compute solutions on the same sheared systems and hence all of them will be able to decide the multiplicities on the intersections.

6.2.2 Decomposing running times

The following paragraphs demonstrate the decomposition of computing-time required by each algorithm in its respective major function calls as these timings were measured in the test-bed polynomial systems. Table 5 presents detailed statistics of every algorithm on every polynomial system from the test-set, while table 4 tries to capture the basic statistical properties of the previous table.

The major function calls and thereby the decomposition of running times and the respective entries on the above tables can be summarized as follows. `Projections` shows the time

system	Average Time		speedup
	M_RUR	G_RUR	
R_1	9	5	1.80
R_2	21	36	0.58
R_3	2	1	2.00
M_1	72	10	7.20
M_2	5	4	1.25
M_3	782	110	7.11
M_4	389	210	1.85
D_1	12	6	2.00
D_2	147	128	1.15
C_1	954	222	4.30
C_2	234	18	13.00
C_3	$1,815$	75	24.20
C_4	$80,650$	370	217.97
C_5	$60,832$	$3,877$	15.69
W_1	$2,115$	247	8.56
W_2	283	114	2.48
W_3	$2,333$	24	97.21
W_4	$77,207$	280	275.74

Table 3: The performance of M_RUR and G_RUR implementations on bivariate solving and the speedup that is achieved when choosing G_RUR.

for the computation of the resultants, `Univ. Solving` for real solving the resultants, and `Sorting` for sorting solutions. In GRID's and M_RUR's case, `biv. solving` corresponds to matching. In G_RUR's case timings for matching are divided between `rational biv.` and $\mathbb{R}_{alg}$ `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 and does not take into account the polynomial system C_5 where GRID failed to reply within 20 minutes. M_RUR spends about 45-50% of its time in matching and about 24-27% in the pre-computation filtering technique. G_RUR spends 55-80% of its time in matching, including gcd computations in an extension field.

Note also the significance of table 5 in order to draw further conclusions regarding the current implementation. Table 4 provides a mean of around 6% for the computation of the **StHa** sequence of f and g required by M_RUR. However, we can observe that this step

	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

Table 4: Statistics on the performance of SLV's algorithms in bivariate solving.

might very well take up to 38.23% of the total computing time. Indeed, a closer look on table 5 reveals that this is the case for the difficult system C_5. Moreover, by table 1 we can observe that M_RUR requires about 61 seconds to solve system C_5. Hence, we can obtain a practical lower bound of about 23 seconds for M_RUR in this case, which is already bad compared to the performance of G_RUR for the entire problem (solving the system). This is a consequence and also a reminder for future work on the implementation of optimal algorithms on subresultant and Sturm-Habicht sequences. As a very important sidenote it should be stressed that implementing these optimal algorithms in sequences computations, the overall performance results for *all* solvers will be improved since the entire library is based on Sturm sequences to perform computations, such as pure univariate solving (root isolation), comparison of real algebraic numbers, and univariate and bivariate sign determination of functions evaluated respectively over one or a pair of real algebraic numbers.

System	GRID			M_RUR							G_RUR				
	Projections	**Univariate**	**Bivariate**	**Projection on x-axis**	**Univariate**	**StHa Sequence**	**Interm. Points**	**Filtering on x-axis**	**Compute K**	**FIND (Biv. Sol.)**	**Projections**	**Univariate**	**Interm. Points**	**Rational Bivariate**	$\mathbb{R}_{alg}$ **Bivariate**
R_1	0.19	73.71	25.78	0.06	28.30	17.91	0.64	1.21	19.79	32.09	0.22	53.75	2.08	43.71	0.02
R_2	0.01	4.47	95.52	0.00	16.30	0.61	0.09	72.84	3.50	6.66	0.07	7.99	0.12	0.10	91.72
R_3	0.53	78.46	20.84	0.17	33.04	20.01	0.97	2.79	27.45	15.57	0.67	40.29	1.85	57.07	0.04
M_1	0.04	10.13	89.75	0.05	21.06	1.46	0.14	35.63	2.97	38.69	0.14	79.62	2.83	16.13	0.02
M_2	0.13	56.29	42.45	0.12	32.57	9.49	3.23	0.68	34.37	19.54	0.48	39.83	3.81	55.07	0.00
M_3	0.00	4.98	95.02	0.02	7.39	0.16	0.02	60.60	1.18	30.62	0.03	28.60	0.67	0.50	70.14
M_4	0.06	99.75	0.19	0.74	91.37	0.44	0.00	1.25	4.43	1.77	0.07	99.37	0.03	0.54	0.00
D_1	0.11	95.25	4.61	0.06	33.81	9.47	0.20	21.14	19.57	15.75	1.20	81.26	0.54	16.93	0.00
D_2	0.01	3.80	96.18	0.00	15.55	0.31	0.11	57.51	1.99	24.53	0.02	17.94	0.22	0.07	81.69
C_1	0.04	2.69	97.27	0.27	5.02	2.37	0.04	28.19	2.02	62.09	0.23	21.00	0.16	2.32	76.25
C_2	0.02	6.60	93.32	0.01	9.40	0.44	0.08	20.57	2.04	67.46	0.22	75.83	2.47	21.08	0.01
C_3	0.01	2.88	97.03	0.04	2.05	1.17	0.00	28.66	1.62	66.46	0.33	16.47	0.16	14.83	67.69
C_4	0.18	2.07	97.74	0.02	0.18	0.08	0.00	1.30	0.09	98.32	0.55	33.57	0.32	3.23	62.00
C_5	–	–	–	0.75	1.92	38.23	0.00	6.43	1.49	51.17	3.89	30.43	0.02	0.35	65.30
W_1	0.04	2.67	97.27	0.07	3.60	1.03	0.02	26.68	1.47	67.13	0.04	20.56	0.16	1.66	77.55
W_2	0.00	7.08	92.89	0.00	11.02	0.22	0.18	39.44	1.72	47.42	0.03	21.78	0.27	0.95	76.89
W_3	0.02	2.18	97.73	0.05	1.63	0.94	0.00	22.26	1.27	73.84	0.41	48.02	3.69	46.37	0.00
W_4	0.01	2.05	97.93	0.00	0.23	0.12	0.00	1.36	0.10	98.19	0.02	33.85	0.51	5.17	60.18

Table 5: Analyzing the percent of time required for various procedures in each algorithm. Values in M_RUR refer to sheared systems (whenever it was necessary). A column about `Sorting` in the case of GRID and G_RUR is not shown.

6.2.3 The effect of filtering

In the following paragraphs we measure the effect of interval arithmetic filters.

GRID Table 6 presents running times for GRID solver in cases where no filtering is performed in computations, i.e. all computations rely on Sturm sequences, or all filters have been applied as these were described in section 6.1. The final column `speedup` indicates the speedup achieved by filters in every case. Based on the numbers of the above table, the

system	deg		sols	Average Time (msecs)		Speedup
				SLV-GRID		
	f	g		NO FILTERS	FILTERED	
R_1	3	4	2	5	5	1.00
R_2	3	1	1	41	66	0.62
R_3	3	1	1	1	1	1.00
M_1	3	3	4	22	87	0.25
M_2	4	2	3	4	4	1.00
M_3	6	3	5	$1,231$	803	1.53
M_4	9	10	2	262	218	1.20
D_1	4	5	1	6	6	1.00
D_2	2	2	4	583	667	0.87
C_1	7	6	6	$2,601$	$1,896$	1.37
C_2	4	3	6	65	177	0.37
C_3	8	7	13	106	580	0.18
C_4	8	7	17	$35,168$	$5,903$	5.98
C_5	16	15	17	$> 20'$	$> 20'$	$-$
W_1	7	6	9	$2,895$	$2,293$	1.26
W_2	4	3	5	514	367	1.40
W_3	8	7	13	104	518	0.20
W_4	8	7	17	$35,054$	$5,410$	6.48

Table 6: Performance averages over 10 runs in MAPLE 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

average speedup achieved by filtering techniques is about 1.51. However, in terms of total computing time for the entire test-set we can observe that:

- Total time without filtering $= 78,662$ msecs.
- Total time with filtering $= 19,001$ msecs.

Hence, the speedup achieved for the entire test-set is about 4.14. Note that in both of the above computations system C_5 has been excluded since neither variation of GRID was able to

solve the system within 20 minutes. However, there are indications that filtering techniques help more in other cases, see for example section 6.4.3.

M_RUR The effect of filtering techniques in the case of M_RUR will be discussed in section 6.4.3 where all solvers deal with bivariate systems in generic position.

G_RUR A similar table with that in the case of GRID is table 7. This time the average

system	deg		sols	Average Time (msecs)		Speedup
				SLV-G_RUR		
	f	g		NO FILTERS	FILTERED	
R_1	3	4	2	6	5	1.20
R_2	3	1	1	36	36	1.00
R_3	3	1	1	1	1	1.00
M_1	3	3	4	10	10	1.00
M_2	4	2	3	4	4	1.00
M_3	6	3	5	141	110	1.28
M_4	9	10	2	201	210	0.96
D_1	4	5	1	6	6	1.00
D_2	2	2	4	171	128	1.34
C_1	7	6	6	236	222	1.06
C_2	4	3	6	18	18	1.00
C_3	8	7	13	75	75	1.00
C_4	8	7	21*	382	370	1.03
C_5	16	15	17	$3,861$	$3,877$	1.00
W_1	7	6	9	277	247	1.12
W_2	4	3	5	141	114	1.23
W_3	8	7	13	24	24	1.00
W_4	8	7	17	318	280	1.13

Table 7: Performance averages over 10 runs in MAPLE 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

speedup achieved by filtering is about 1.08. In terms of total computing time for the entire test-set we can observe that:

- Total time without filtering $= 5,908$ msecs.
- Total time with filtering $= 5,737$ msecs.

In other words, the speedup that is achieved by filtering for the entire test-set is about 1.03. Thus G_RUR seems not to be affected at a significant level by filtering. However,

this is more or less expected since G_RUR relies heavily on gcd computations in extension fields and MAPLE's built-in function for factoring. Even when computing the multiplicities of the given system, G_RUR seems not to be affected much from filtering. For a more concrete comparison, please refer to section 6.4.3 that discusses the problem of computing the multiplicities of the given system.

6.3 Bivariate solving and other packages

For the sake of completeness on the evaluation of the initial release of the SLV library tests have been made with other solvers on the same polynomial systems. First of all, FGB/RS [2] [Rou99], which performs exact real solving using Gröbner bases and RUR, through its MAPLE interface has been tested. It should be underlined though that communication with MAPLE increases the runtimes and additional tuning might offer 20-30% efficiency increase. Moreover, 3 SYNAPS [3] solvers have been tested: STURM is a naive implementation of GRID [ET05]; SUBDIV implements [MP05], and is based on Bernstein basis and `double` arithmetic. It needs an initial box for computing the real solutions of the system and in all the cases the box $[-10, 10] \times [-10, 10]$ was used. NEWMAC [MT00], is a general purpose solver based on computations of generalized eigenvectors using LAPACK, which computes all complex solutions.

Other MAPLE implementations have also been tested: INSULATE is a package that implements [WS05] for computing the topology of real algebraic curves, and TOP implements [GVN02]. Both packages were kindly provided 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 and hence achieve an accurate timing in every case. Finally, it should be noted that TOP has an additional parameter that sets the initial precision (decimal 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 [Ker06] and recorded its performance on initial values of 60 and 500 digits.

It should be underlined that experiments are not considered as competition, but as 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 MAPLE is hard, since it is a general purpose package and a lot of overhead is added to its function calls. For example this is the case for FGB/RS.

Overall performance results are shown on tab. 8, averaged over 10 iterations. Although the current solver of choice for SLV library is G_RUR, the other solvers are presented as well for completeness. Note that for the first data set, there are no timings for INSULATE and TOP since it was not easy to modify their code so as to deal with general polynomial systems. The rest (systems C_i and W_i) correspond to algebraic curves, i.e. polynomial systems of the form $f = \frac{\partial f}{\partial y} = 0$, that all packages can deal with.

[2] `http://www-spaces.lip6.fr/index.html`
[3] `http://www-sop.inria.fr/galaad/logiciels/synaps/`

In cases where the solvers failed to find the correct number of real solutions we indicate so with an asterisk (*). In the case of NEWMAC where all complex solutions are computed, the (*) is placed in one more case: since NEWMAC computes all complex solutions, a further computing step is required so as to distinguish the ones that reflect the real solutions.

system	deg		solutions	Average Time (msecs)									
				BIVARIATE SOLVING							TOPOLOGY		
				SLV			FGB/RS	SYNAPS			INSULATE	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 8: Performance averages over 10 runs in MAPLE 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

6.3.1 G_RUR and other solvers

In the following paragraphs we will try to compare the performance of G_RUR with the rest of the solvers. For this purpose, we conduct speedup-tables like the ones that were drawn in section 6.2.1.

G_RUR vs. FGB/RS Table 9 presents running times for FGB/RS and G_RUR as well as the speedup that one gains when choosing G_RUR instead of FGB/RS for bivariate solving. As it is shown from the table G_RUR is faster than FGB/RS in 8 out of the 18 instances,

system	Average Time		speedup
	FGB/RS	G_RUR	
R_1	26	5	5.20
R_2	24	36	0.67
R_3	22	1	22.00
M_1	25	10	2.50
M_2	24	4	6.00
M_3	30	110	0.27
M_4	158	210	0.75
D_1	28	6	4.67
D_2	26	128	0.20
C_1	93	222	0.42
C_2	27	18	1.50
C_3	54	75	0.72
C_4	138	370	0.37
C_5	$4,044$	$3,877$	1.04
W_1	92	247	0.37
W_2	29	114	0.25
W_3	56	24	2.33
W_4	148	280	0.53

Table 9: The performance of FGB/RS and G_RUR on bivariate solving and the speedup that is achieved when choosing G_RUR.

including the difficult system C_5. The speedup factor ranges from 0.2 to 22 with an average of 2.62. However, in terms of total computing times for the entire test-set we can observe that:

- Total time for FGB/RS $= 5,044$ msecs.
- Total time for G_RUR $= 5,737$ msecs.

Hence, the speedup in terms of total computing time is about 0.88. This is an indication that although the computation of the ideal of the given system is a more expensive operation on average, it may be faster when someone faces a set of different polynomial systems.

G_RUR VS. SYNAPS/STURM Let's move on with a comparison between G_RUR and SYNAPS's STURM implementation. Table 10 presents running times for SYNAPS/STURM and G_RUR and the speedup gained when preferring G_RUR. G_RUR is faster than STURM in

system	Average Time		speedup
	STURM	G_RUR	
R_1	2	5	0.40
R_2	1	36	0.03
R_3	1	1	1.00
M_1	2	10	0.20
M_2	1	4	0.25
M_3	230	110	2.09
M_4	90	210	0.43
D_1	2	6	0.33
D_2	21	128	0.16
C_1	479	222	2.16
C_2	12	18	0.67
C_3	23	75	0.31
C_4	$3,495$	370	9.45
C_5	$> 20'$	$3,877$	$-$
W_1	954	247	3.86
W_2	20	114	0.18
W_3	32	24	1.33
W_4	$4,086$	280	14.59

Table 10: The performance of SYNAPS/STURM and G_RUR on bivariate solving and the speedup that is achieved when choosing G_RUR.

6 out of the 18 instances. On the other hand, G_RUR behaves worse usually in polynomial systems that are solved by both implementations in less than 100 msecs, something that is expected since STURM is implemented in C++. However, as the dimension of the polynomial systems increases, G_RUR outperforms STURM and the latter's lack of modular algorithms for computing resultants is more and more evident. Overall, an average speedup of about 2.2 is achieved when someone prefers G_RUR. In terms of total computing times for the entire test-set (excluding system C_5 where STURM failed to reply within 20 minutes) we can observe that:

- Total time for SYNAPS/STURM $= 9,451$ msecs.

- Total time for G_RUR $= 1,860$ msecs.

Hence, if someone considers the speedup that is achieved in terms of total computing time for the entire test set, it can be observed that G_RUR is about 5.08 times faster than STURM highlighting the previous remark regarding resultants in SYNAPS.

G_RUR VS. SYNAPS/SUBDIV We now switch to a comparison between G_RUR and SYNAPS's SUBDIV implementation. Table 11 presents running times for SYNAPS/SUBDIV and G_RUR, and as in the earlier tables, the last column shows the speedup gained when preferring G_RUR. It should be mentioned however, that SUBDIV requires an initial box where all the real solutions of the system reside. In the experiments, the `box` $[-10, 10] \times [-10, 10]$ was used in every case. The solver was called with the following command:

```
sols = solve( pols, SBDSLV< NT, SBDSLV_RDL >(1e-10), box);
```

where `pols` are of type `Seq< MPOL >` and `sols` are of type `Seq< VectDse< NT> >`. Finally, in cases where SUBDIV failed to compute the correct number of real solutions, an asterisk (*) is placed to indicate so. G_RUR is faster than SUBDIV in half of the instances. However, the case is similar to STURM's. G_RUR may require more computing time on polynomial systems that are solved in less than 400 msecs by both solvers, while on system C_5 G_RUR is faster than SUBDIV by about 2.47 seconds. A striking experimental result though is SUBDIV's inefficiency on polynomial systems C_1 and W_1. Note that the initial box $[-10, 10] \times [-10, 10]$ is not large enough to justify easily such deficiency. For example, all real solutions of the system C_1 can be found inside the rectangle $[-2, 2] \times [-1, 2]$ while the x-coordinates can take both of the extreme values; i.e. -2 and 2. On average, G_RUR achieves a speedup of 62.92 which is the result of the problematic behavior of SUBDIV in systems C_1 and W_1. If these systems are omitted from the computation, then G_RUR achieves a speedup of 8.93. In terms of total computing times for the entire test-set we can observe that:

- Total time for SYNAPS/SUBDIV $= 238,255$ msecs.
- Total time for G_RUR $= 5,737$ msecs.

Hence, the speedup under these terms is about 41.53 favoring G_RUR. However, this value is again greatly increased due to systems C_1 and W_1. Omitting these systems, we can observe that total computing times are as follows:

- Total time for SYNAPS/SUBDIV $= 12,950$ msecs.
- Total time for G_RUR $= 5,268$ msecs.

This time the speedup in terms of total computing time is about 2.46. As a final comment, one can not forget that SUBDIV is based on finite precision arithmetic and consequently numerical errors occur in the computations of the solutions as this is signified by an asterisk (*) in tables 8 and 11.

system	Average Time		speedup
	SUBDIV	G_RUR	
R_1	2	5	0.40
R_2	1	36	0.03
R_3	2	1	2.00
M_1	1	10	0.10
M_2	289*	4	72.25
M_3	5,058*	110	45.98
M_4	3*	210	0.01
D_1	5	6	0.83
D_2	1*	128	0.01
C_1	170,265*	222	766.96
C_2	23*	18	1.28
C_3	214*	75	2.85
C_4	217*	370	0.59
C_5	6,345*	3,877	1.64
W_1	55,040*	247	222.83
W_2	224*	114	1.96
W_3	285*	24	11.88
W_4	280*	280	1.00

Table 11: The performance of SYNAPS/SUBDIV and G_RUR on bivariate solving and the speedup that is achieved when choosing G_RUR.

G_RUR VS. SYNAPS/NEWMAC A comparison between G_RUR and SYNAPS's NEWMAC implementation can be made with the help of table 12 which has similar structure with the previous tables. The solver was called with the following command:

```
sols = solve( pols, Newmac<coeff_t,sol_t>());
```

where `pols` are of type `std::list<MPOL>` and `sols` are of type `sol_t`. Note that an asterisk (*) indicates incorrect number of real solutions. The problem is that NEWMAC computes all *complex* solutions of the input polynomial system and some considerations are needed in these cases. But these will be addressed in the following paragraph. G_RUR is faster than NEWMAC in systems M_4, D_1 and W_3 and exhibits similar performance in systems R_1 and R_3. This time the average speedup is about 0.53 if someone prefers G_RUR, and in terms of total computing times for the entire test-set we have:

- Total time for SYNAPS/NEWMAC $= 1,353$ msecs.
- Total time for G_RUR $= 5,737$ msecs.

system	Average Time		speedup
	NEWMAC	G_RUR	
R_1	5*	5	1.00
R_2	1*	36	0.03
R_3	1*	1	1.00
M_1	2*	10	0.20
M_2	2*	4	0.50
M_3	7*	110	0.06
M_4	447*	210	2.13
D_1	8*	6	1.33
D_2	2	128	0.02
C_1	39*	222	0.18
C_2	4*	18	0.22
C_3	25*	75	0.33
C_4	190*	370	0.51
C_5	346*	$3,877$	0.09
W_1	39*	247	0.16
W_2	3*	114	0.03
W_3	25*	24	1.04
W_4	207*	280	0.74

Table 12: The performance of SYNAPS/NEWMAC and G_RUR on bivariate solving and the speedup that is achieved when choosing G_RUR.

In other words, G_RUR is slower than newmac about 4.24 times for the entire test-set.

However, these numbers do not necessarily reflect the truth for various reasons. First of all, NEWMAC is based on computations of generalized eigenvectors using LAPACK, which computes all complex solutions. This can be really fast in practice, but an additional problem arises; that of classifying the real solutions among all complex solutions computed by NEWMAC. This is not as trivial as it may sound, since finite precision arithmetic is used, resulting in numerical errors while computing all complex solutions. So, there is one problem on retracting only the real solutions among all complex solutions computed (with the possible numerical errors that these may contain). In addition to that, finite precision has further impacts on the solution set that is computed. There are cases where NEWMAC may not compute some of the real solutions. A representative example in this class of problems is system C_4 which has 17 real solutions and NEWMAC claims that the total number of real and complex solutions is exactly 0.

Hence, NEWMAC requires a better and more accurate implementation than LAPACK when computing the various eigenvectors and eigenvalues that are needed in order to solve the input systems. However, this might still not eliminate all numerical errors that are introduced

in the entire complex solution set. Even with this enhancement, some additional time will be possibly required in order to filter the few (in general) real solutions among the entire complex solution set. For instance, NEWMAC computes 90 complex solutions in system M_4 while the number of real solutions for the system is only 2. Concluding, having all these observations in mind, G_RUR seems to be a competitive alternative to NEWMAC since it is not affected by these problems.

G_RUR VS. INSULATE Let's turn our attention on a comparison between G_RUR and INSULATE which computes the topology of real plane algebraic curves. In this case INSULATE has been modified so as to stop as soon as it computes all real solutions. A comparative performance table similar to the ones in the previous paragraphs is presented in table 13. Note that the comparison takes place on the second set of the test-set where the polynomial systems are of the form $f = \frac{\partial f}{\partial y} = 0$ that both packages can manage. G_RUR is faster in all

system	Average Time		speedup
	INSULATE	G_RUR	
C_1	524	222	2.36
C_2	28	18	1.55
C_3	327	75	4.36
C_4	$1,589$	370	4.29
C_5	$179,182$	$3,877$	46.22
W_1	517	247	2.09
W_2	27	114	0.24
W_3	309	24	12.88
W_4	$1,579$	280	5.64

Table 13: The performance of INSULATE and G_RUR on bivariate solving and the speedup that is achieved when choosing G_RUR.

but W_2 system yielding an average speedup this time of 8.85. However, as the dimension of the input polynomial systems increases, G_RUR seems to be more efficient. In terms of total computing time for the entire test set we can observe:

- Total time for INSULATE $= 184,082$ msecs.
- Total time for G_RUR $= 5,227$ msecs.

Hence the speedup under this point of view is about 35.22. In any case though, the amount of experiments is relatively small in order to draw safe conclusions on the relative performance of the two implementations in real solving of bivariate polynomial systems.

G_RUR VS. TOP Finally, a comparison between G_RUR and TOP which computes the topology of real plane algebraic curves is performed. Recall that TOP requires an extra

parameter which sets the initial precision in computations (decimal digits). As it has already been stated, this is a problem since there is no easy way on computing a good value and furthermore, a very low initial precision might result in loss in the number of real solutions, while a very high initial precision might result in performance deficiency. For this purpose, the route of [Ker06] has been followed and the performance of TOP was recorded for initial precisions of 60 and 500 digits. Similarly with INSULATE case, the comparison takes place on the systems C_i and W_i that are of the form $f = \frac{\partial f}{\partial y} = 0$ that both packages can manage.

60 **digits precision:** The comparison in this case is shown in table 14. G_RUR is

system	Average Time		speedup
	TOP$_{60}$	G_RUR	
C_1	409	222	1.84
C_2	36	18	3.00
C_3	693	75	9.24
C_4	$1,624$	370	4.39
C_5	$91,993$	$3,877$	23.73
W_1	419	247	1.70
W_2	20	114	0.18
W_3	525	24	21.88
W_4	$1,458$	280	5.21

Table 14: The performance of TOP with precision set to 60 digits and G_RUR on bivariate solving and the speedup that is achieved when choosing G_RUR.

faster in all but W_2 system yielding an average speedup this time of 7.79. Similarly with INSULATE's case, as the dimension of the input polynomial systems increases, G_RUR seems to be more efficient. In terms of total computing time for the entire test set we can observe:

- Total time for TOP$_{60}$ $= 97,177$ msecs.
- Total time for G_RUR $= 5,227$ msecs.

Hence the speedup under this point of view is about 18.59.

500 **digits precision:** The comparison in this case is shown in table 15. An interesting result is that although TOP computations have been slowed down with this precision, TOP is still faster in solving system W_2. This time the average speedup that is achieved by G_RUR is 22.64. In terms of total computing time for the entire test set this time we have:

- Total time for TOP$_{500}$ $= 199,491$ msecs.
- Total time for G_RUR $= 5,227$ msecs.

Hence the speedup under this point of view is about 38.17.

system	Average Time		speedup
	TOP$_{500}$	G_RUR	
C_1	1,367	222	6.16
C_2	115	18	6.39
C_3	2,829	75	37.72
C_4	6,435	370	17.39
C_5	180,917	3,877	46.66
W_1	1,350	247	5.47
W_2	60	114	0.53
W_3	1,588	24	66.17
W_4	4,830	280	17.25

Table 15: The performance of TOP with precision set to 500 digits and G_RUR on bivariate solving and the speedup that is achieved when choosing G_RUR.

6.4 Computing multiplicities

This section presents the performance of SLV library when someone wants to compute the multiplicities at the various intersecting points. Moreover, the effect of filtering will be discussed once more.

Overall performance results for the three projection based algorithms are shown on table 16. In order to compute the multiplicities the initial systems were sheared whenever it was necessary based on the algorithm that was presented in section 4.1.1. Since the polynomial systems were in generic position, the algorithms stopped searching for solution along the various vertical lines as soon as a solution was computed. Note that running times in M_RUR's case have not changed from table 1 since M_RUR by default requires a system in generic position.

Once again G_RUR presents the best performance. It is faster in 17 out of the 18 instances and apart from system C_5 provides solutions in less than a second. Moreover, now that the sheared systems have little or no linear factors and slightly increased bitsize GRID's high complexity starts to become more apparent: M_RUR is faster in 10 out of the 18 instances. In addition to that, it should be stressed once again that M_RUR's inefficiency is basically due to the lack of optimal algorithms for computing the various Sturm sequences. For example, when solving system C_5 M_RUR requires more than 23 seconds simply to generate the **StHa** sequence of the input polynomials f and g.

6.4.1 Comparing SLV solvers

The following paragraphs will briefly compare G_RUR with GRID and M_RUR when computing multiplicities. Moreover, this time a comparison between M_RUR and GRID will be performed.

system	deg		$\mathbb{R}_{\mathbf{alg}}$	Average Time (msecs)		
	f	g	solutions	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

Table 16: Performance averages over 10 runs in MAPLE 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

G_RUR VS. GRID Table 17 presents running times for GRID and G_RUR when computing multiplicities. Again, the final column indicates the speedup that is achieved when someone prefers G_RUR. As it is shown from the table 17 G_RUR can be up to 15.81 times faster than GRID with an average speedup of around 5.26 among the input systems and excluding system C_5 where GRID failed to reply within 20 minutes. Moreover, in terms of total computing times for the entire test-set (again excluding system C_5) we can observe that:

- Total time for GRID $= 28,272$ msecs.
- Total time for G_RUR $= 3,313$ msecs.

In other words, the speedup in terms of total computing time is about 8.53.

G_RUR VS. M_RUR Table 18 presents running times for M_RUR and G_RUR when computing multiplicities. Similarly with the previous table, the final column indicates the speedup that is achieved when preferring G_RUR. This time G_RUR can be up to 170.15 times faster than M_RUR with an average speedup of around 18.77 among the input poly-

system	Average Time		speedup
	GRID	G_RUR	
R_1	6	6	1.00
R_2	66	36	1.83
R_3	1	1	1.00
M_1	183	45	4.07
M_2	4	4	1.00
M_3	$4,871$	393	12.39
M_4	339	199	1.70
D_1	6	6	1.00
D_2	567	126	4.50
C_1	$1,702$	247	6.89
C_2	400	99	4.04
C_3	669	152	4.40
C_4	$7,492$	474	15.81
C_5	$> 20'$	$6,367$	$-$
W_1	$3,406$	393	8.67
W_2	$1,008$	193	5.22
W_3	$1,769$	230	7.69
W_4	$5,783$	709	8.16

Table 17: The performance of GRID and G_RUR implementations when computing multiplicities on the intersections and the speedup that is achieved when choosing G_RUR.

nomial systems. Moreover, in terms of total computing times for the entire test-set we can observe that:

- Total time for M_RUR $= 227,862$ msecs.
- Total time for G_RUR $= 9,680$ msecs.

In other words, the speedup in terms of total computing time is about 23.54.

M_RUR vs. GRID Table 19 presents running times for GRID and G_RUR when computing multiplicities. The final column in this table indicates the speedup that is achieved when preferring M_RUR for this operation. Excluding system C_5 where GRID failed to reply within 20 minutes, M_RUR can be up to 6.23 times faster, yielding an average speedup of around 1.71 among the input systems. Moreover, in terms of total computing times for the entire test-set (again excluding system C_5) we can observe that:

- Total time for GRID $= 28,272$ msecs.
- Total time for G_RUR $= 167,030$ msecs.

system	Average Time		speedup
	M_RUR	G_RUR	
R_1	9	6	1.50
R_2	21	36	0.58
R_3	2	1	2.00
M_1	72	45	1.60
M_2	5	4	1.25
M_3	782	393	1.99
M_4	389	199	1.95
D_1	12	6	2.00
D_2	147	126	1.17
C_1	954	247	3.86
C_2	234	99	2.36
C_3	$1,815$	152	11.94
C_4	$80,650$	474	170.15
C_5	$60,832$	$6,367$	9.55
W_1	$2,115$	393	5.38
W_2	283	193	1.47
W_3	$2,333$	230	10.14
W_4	$77,207$	709	108.90

Table 18: The performance of M_RUR and G_RUR implementations when computing multiplicities on the intersections and the speedup that is achieved when choosing G_RUR.

In other words, the speedup in terms of total computing time is about 0.17. However, it should be mentioned once again that system C_5 is not considered in these values.

6.4.2 Decomposing running times

This section is similar to 6.2.2. It presents statistics for the various solvers in the *sheared* case of the test-set polynomial systems. Hence, the interpretation of the two tables is identical to the tables presented in section 6.2.2.

Things have not changed much from section 6.2.2 in GRID's and M_RUR's case. In a nutshell, GRID spends more than 72% of its time in matching. Similarly with table 4, this percent includes the application of filters and does not take into account the polynomial system C_5 where GRID failed to reply within 20 minutes. M_RUR spends about 45-50% of its time in matching and about 24-27% in the pre-computation filtering technique. Finally, G_RUR spends 68-80% of its time in matching, including gcd computations in an extension field. This time, in absence of excessive factoring G_RUR spends significantly more time in bivariate solving.

system	Average Time		speedup
	GRID	M_RUR	
R_1	6	9	0.67
R_2	66	21	3.14
R_3	1	2	0.50
M_1	183	72	2.54
M_2	4	5	0.80
M_3	$4,871$	782	6.23
M_4	339	389	0.87
D_1	6	12	0.50
D_2	567	147	3.86
C_1	$1,702$	954	1.78
C_2	400	234	1.71
C_3	669	$1,815$	0.37
C_4	$7,492$	$80,650$	0.09
C_5	$> 20'$	$60,832$	$-$
W_1	$3,406$	$2,115$	1.61
W_2	$1,008$	283	3.56
W_3	$1,769$	$2,333$	0.76
W_4	$5,783$	$77,207$	0.07

Table 19: The performance of GRID and M_RUR implementations when computing multiplicities on the intersections and the speedup that is achieved when choosing M_RUR.

The equivalent table to table 5 is table 21. It presents the running-time breakdown for the various algorithms in the various cases. Again note, that values presented in M_RUR's case are identical in both tables due to the requirements of the algorithm, i.e. M_RUR has to solve sheared systems.

	phase of the algorithm	interval		median	mean	std dev
		min	max			
GRID	projections	00.00	00.53	00.06	00.08	00.12
	univ. solving	01.65	99.63	05.42	27.39	37.65
	biv. solving	00.30	98.33	96.75	72.42	37.82
	sorting	00.00	01.15	00.02	00.11	00.27
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	03.73	00.11	00.58	01.14
	univ. solving	06.60	99.16	22.35	30.27	23.48
	inter. points	00.01	03.93	00.20	00.59	01.05
	rational biv.	00.07	55.59	02.61	11.91	19.22
	$\mathbb{R}_{alg}$ biv.	00.00	93.04	77.51	56.50	35.53
	sorting	00.00	00.83	00.08	00.14	00.21

Table 20: Statistics on the performance of SLV's algorithms when computing multiplicities.

System	GRID			M_RUR							G_RUR				
	Projections	**Univariate**	**Bivariate**	**Projection on x-axis**	**Univariate**	**StHa Sequence**	**Interm. Points**	**Filtering on x-axis**	**Compute K**	**FIND (Biv. Sol.)**	**Projections**	**Univariate**	**Interm. Points**	**Rational Bivariate**	**$\mathbb{R}_{alg}$ Bivariate**
R_1	0.11	84.14	15.40	0.06	28.30	17.91	0.64	1.21	19.79	32.09	0.30	45.69	1.08	52.68	0.00
R_2	0.01	4.26	95.73	0.00	16.30	0.61	0.09	72.84	3.50	6.66	0.04	6.60	0.10	0.21	93.04
R_3	0.53	82.86	16.43	0.17	33.04	20.01	0.97	2.79	27.45	15.57	0.57	40.79	2.94	55.59	0.04
M_1	0.00	7.94	92.04	0.05	21.06	1.46	0.14	35.63	2.97	38.69	0.03	25.53	0.40	5.57	68.39
M_2	0.14	60.80	37.92	0.12	32.57	9.49	3.23	0.68	34.37	19.54	3.73	38.23	3.93	53.28	0.00
M_3	0.01	1.66	98.33	0.02	7.39	0.16	0.02	60.60	1.18	30.62	0.02	12.38	0.09	0.31	87.20
M_4	0.06	99.63	0.30	0.74	91.37	0.44	0.00	1.25	4.43	1.77	0.26	99.16	0.03	0.55	0.00
D_1	0.11	95.32	4.54	0.06	33.81	9.47	0.20	21.14	19.57	15.75	1.20	81.22	0.60	16.91	0.00
D_2	0.01	4.13	95.86	0.00	15.55	0.31	0.11	57.51	1.99	24.53	0.02	17.96	0.22	0.07	81.67
C_1	0.09	2.82	97.09	0.27	5.02	2.37	0.04	28.19	2.02	62.09	0.05	17.60	0.15	2.61	79.54
C_2	0.01	5.42	94.54	0.01	9.40	0.44	0.08	20.57	2.04	67.46	0.03	22.35	0.33	2.35	74.40
C_3	0.02	4.71	95.23	0.04	2.05	1.17	0.00	28.66	1.62	66.46	0.06	21.66	0.12	10.70	67.25
C_4	0.18	1.65	98.16	0.02	0.18	0.08	0.00	1.30	0.09	98.32	0.27	26.36	0.11	2.53	70.62
C_5	–	–	–	0.75	1.92	38.23	0.00	6.43	1.49	51.17	3.69	20.07	0.01	0.27	75.95
W_1	0.03	2.16	97.79	0.07	3.60	1.03	0.02	26.68	1.47	67.13	0.11	18.79	0.09	1.64	79.32
W_2	0.00	3.25	96.75	0.00	11.02	0.22	0.18	39.44	1.72	47.42	0.02	16.27	0.14	1.05	82.47
W_3	0.02	1.98	97.98	0.05	1.63	0.94	0.00	22.26	1.27	73.84	0.04	13.55	0.14	6.58	79.57
W_4	0.02	2.86	97.11	0.00	0.23	0.12	0.00	1.36	0.10	98.19	0.09	20.60	0.20	1.54	77.51

Table 21: Analyzing the percent of time required for various procedures in each algorithm. All values refer to the sheared systems (whenever it was necessary). A column about `Sorting` in the case of GRID and G_RUR is not shown.

6.4.3 The effect of filtering

Similarly with section 6.2.3 this section examines the effect of filtering techniques on the performance of all solvers.

GRID Table 22 presents running times for GRID solver in cases where no filtering is performed in computations, i.e. all computations rely on Sturm sequences, or all filters have been applied as these were described in section 6.1. The final column `speedup` indicates the speedup achieved by filters in every case. Based on the numbers of the above table, the

system	deg		sols	Average Time (msecs)		Speedup
				SLV-GRID		
	f	g		NO FILTERS	FILTERED	
R_1	3	4	2	4	6	0.67
R_2	3	1	1	40	66	0.61
R_3	3	1	1	1	1	1.00
M_1	3	3	4	172	183	0.94
M_2	4	2	3	4	4	1.00
M_3	6	3	5	$118,215$	$4,871$	24.27
M_4	9	10	2	404	339	1.19
D_1	4	5	1	6	6	1.00
D_2	2	2	4	418	567	0.74
C_1	7	6	6	$5,162$	$1,702$	3.03
C_2	4	3	6	464	400	1.16
C_3	8	7	13	155	669	0.23
C_4	8	7	17	$27,126$	$7,492$	3.62
C_5	16	15	17	$> 20'$	$> 20'$	$-$
W_1	7	6	9	$10,091$	$3,406$	2.96
W_2	4	3	5	$1,508$	$1,008$	1.50
W_3	8	7	13	$1,338$	$1,769$	0.76
W_4	8	7	17	$50,808$	$5,783$	8.79

Table 22: Performance averages over 10 runs in MAPLE 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

average speedup achieved by filtering techniques is about 3.14. However, in terms of total computing time for the entire test-set we can observe that:

- Total time without filtering $= 215,916$ msecs.
- Total time with filtering $= 28,272$ msecs.

Hence, the speedup achieved for the entire test-set is about 7.64. Note that in both of the above computations system C_5 has been excluded since neither variation of GRID was able to solve the system within 20 minutes.

M_RUR Table 23 presents the performance of the M_RUR solver with the application of all filters or not. Recall, that M_RUR uses one more heuristic technique (refer to section 6.1). This heuristic was present in the running times that are shown in filtered case in table 23. M_RUR was unable to solve system C_5 within 20 minutes when filtering techniques were

system	deg		sols	Average Time (msecs)		Speedup
				SLV-M_RUR		
	f	g		NO FILTERS	FILTERED	
R_1	3	4	2	9	9	1.00
R_2	3	1	1	8	21	0.38
R_3	3	1	1	2	2	1.00
M_1	3	3	4	49	72	0.68
M_2	4	2	3	4	5	0.80
M_3	6	3	5	2,054	782	2.63
M_4	9	10	2	323	389	0.83
D_1	4	5	1	10	12	0.83
D_2	2	2	4	88	147	0.60
C_1	7	6	6	22,006	954	23.07
C_2	4	3	6	138	234	0.59
C_3	8	7	13	38,307	1,815	21.11
C_4	8	7	17	784,613	80,650	9.73
C_5	16	15	17	$> 20'$	60,832	–
W_1	7	6	9	45,323	2,115	21.43
W_2	4	3	5	249	283	0.88
W_3	8	7	13	50,724	2,333	21.74
W_4	8	7	17	839,708	77,207	10.88

Table 23: Performance averages over 10 runs in MAPLE 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

not present in the computations. In the rest of the cases, the average speedup achieved by filtering techniques is about 6.95. In terms of total computing time for the entire test-set (again excluding system C_5 from the computations) we can observe that:

- Total time without filtering = 1,783,615 msecs.
- Total time with filtering = 167,030 msecs.

Hence, the speedup achieved for the entire test-set is about 10.68.

The effect of preprocessing x-candidates However, it is interesting to investigate the effect of preprocessing x-candidates on M_RUR's performance. For this purpose, table 24 presents running times when this heuristic technique is applied or not (but interval arithmetic and gcd filtering are applied) and the speedup that is achieved with its application. Hence,

system	deg		sols	Average Time (msecs)		Speedup
				SLV-M_RUR		
	f	g		−Preprocess	+Preprocess	
R_1	3	4	2	10	9	1.11
R_2	3	1	1	10	21	0.48
R_3	3	1	1	2	2	1.00
M_1	3	3	4	64	72	0.89
M_2	4	2	3	5	5	1.00
M_3	6	3	5	591	782	0.76
M_4	9	10	2	290	389	0.75
D_1	4	5	1	10	12	0.83
D_2	2	2	4	126	147	0.86
C_1	7	6	6	$2,672$	954	2.80
C_2	4	3	6	246	234	1.05
C_3	8	7	13	$14,276$	$1,815$	7.87
C_4	8	7	17	$282,798$	$80,650$	3.51
C_5	16	15	17	$> 20'$	$60,832$	–
W_1	7	6	9	$9,239$	$2,115$	4.37
W_2	4	3	5	354	283	1.25
W_3	8	7	13	$13,235$	$2,333$	5.67
W_4	8	7	17	$242,199$	$77,207$	3.14

Table 24: Performance averages over 10 runs in MAPLE 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

the preprocessing heuristic provides M_RUR a speedup of about 2.20 on average. In terms of total computing time for the entire test-set we can observe that:

- Total time without preprocessing $= 566,127$ msecs.
- Total time with preprocessing $= 167,030$ msecs.

In other words, the speedup achieved for the entire test-set due to preprocessing is about 3.39. Note that in both of the above computations system C_5 has been excluded since neither variation of GRID was able to solve the system within 20 minutes.

G_RUR Table 25 presents the performance of the G_RUR solver with the application of filters or not. Based on the numbers of the above table, the average speedup achieved by

system	deg		sols	Average Time (msecs)		Speedup
				SLV-G_RUR		
	f	g		NO FILTERS	FILTERED	
R_1	3	4	2	6	6	1.00
R_2	3	1	1	36	36	1.00
R_3	3	1	1	1	1	1.00
M_1	3	3	4	54	45	1.20
M_2	4	2	3	5	4	1.25
M_3	6	3	5	619	393	1.58
M_4	9	10	2	273	199	1.37
D_1	4	5	1	6	6	1.00
D_2	2	2	4	171	126	1.36
C_1	7	6	6	278	247	1.13
C_2	4	3	6	137	99	1.38
C_3	8	7	13	146	152	0.96
C_4	8	7	17	494	474	1.04
C_5	16	15	17	$8,448$	$6,367$	1.33
W_1	7	6	9	482	393	1.23
W_2	4	3	5	297	193	1.54
W_3	8	7	13	296	230	1.29
W_4	8	7	17	978	709	1.38

Table 25: Performance averages over 10 runs in MAPLE 9.5 on a 2GHz AMD64@3K+ processor with 1GB RAM.

filtering techniques is about 1.22. In terms of total computing time for the entire test-set we have:

- Total time without filtering $= 12,727$ msecs.
- Total time with filtering $= 9,680$ msecs.

Hence, the speedup achieved for the entire test-set is about 1.31. Once again we observe that filtering techniques do not help much G_RUR.

A Experiments

The following list describes the polynomial systems used for testing our implementation.

A.1 Input Polynomials

System R_1:

$$\begin{aligned} f &= 1 + 2x - 2x^2y - 5xy + x^2 + 3x^2y \\ g &= 2 + 6x - 6x^2y - 11xy + 4x^2 + 5x^3y \end{aligned}$$

System R_2:

$$\begin{aligned} f &= x^3 + 3x^2 + 3x - y^2 + 2y - 2 \\ g &= 2x + y - 3 \end{aligned}$$

System R_3:

$$\begin{aligned} f &= x^3 - 3x^2 - 3xy + 6x + y^3 - 3y^2 + 6y - 5 \\ g &= x + y - 2 \end{aligned}$$

System M_1:

$$\begin{aligned} f &= y^2 - x^2 + x^3 \\ g &= y^2 - x^3 + 2x^2 - x \end{aligned}$$

System M_2:

$$\begin{aligned} f &= x^4 - 2x^2y + y^2 + y^4 - y^3 \\ g &= y - 2x^2 \end{aligned}$$

System M_3:

$$\begin{aligned} f &= x^6 + 3x^4y^2 + 3x^2y^4 + y^6 - 4x^2y^2 \\ g &= y^2 - x^2 + x^3 \end{aligned}$$

System M_4:

$$\begin{aligned} f &= x^9 - y^9 - 1 \\ g &= x^{10} + y^{10} - 1 \end{aligned}$$

System D_1:

$$\begin{aligned} f &= x^4 - y^4 - 1 \\ g &= x^5 + y^5 - 1 \end{aligned}$$

System D_2:

$$\begin{aligned} f &= -312960 - 2640x^2 - 4800xy - 2880y^2 + 58080x + 58560y \\ g &= -584640 - 20880x^2 + 1740xy + 1740y + 274920x - 59160y \end{aligned}$$

System C_1:

$$\begin{aligned} f &= (x^3 + x - 1 - xy + 3y - 3y^2 + y^3) \\ &\quad (x^4 + 2x^2y^2 - 4x^2 - y^2 + y^4) \\ g &= \mathrm{diff}(f, y) \end{aligned}$$

System C_2:

$$\begin{aligned} f &= y^4 - 6y^2x + x^2 - 4x^2y^2 + 24x^3 \\ g &= \mathrm{diff}(f, y) \end{aligned}$$

System C_3:

$$\begin{aligned} f = ((x-1)^2 + y^2 - 2)((x+1)^2 + y^2 - 2) \\ ((x-1)^2 + (y+2)^2 - 2)((x+1)^2 + (y+2)^2 - 2) \\ g = \mathrm{diff}(f, y) \end{aligned}$$

System C_4:

$$\begin{aligned} f = (x^2 - 2x - 1 + y^2)(x^2 + 2x - 1 + y^2) \\ (x^2 - 2x + 3 + y^2 + 4y) \\ (100000x^2 + 200000x + 299999 + 100000y^2 + 400000y) \\ g = \mathrm{diff}(f, y) \end{aligned}$$

System C_5:

$$\begin{aligned} f = (x^4 + 4x^3 + 6x^2 + 4x + y^4 + 4y^3 + 6y^2 + 4y) \\ (x^4 + 4x^3 + 6x^2 + 4x + y^4 - 4y^3 + 6y^2 - 4y) \\ (x^4 - 4x^3 + 6x^2 - 4x + y^4 + 4y^3 + 6y^2 + 4y) \\ (100000x^4 - 400000x^3 + 600000x^2 - 400000x \\ -1 + 100000y^4 - 400000y^3 + 600000y^2 - 400000y) \\ g = \mathrm{diff}(f, y) \end{aligned}$$

System W_1:

$$\begin{aligned} f = (x^3 + x - 1 - xy + 3y - 3y^2 + y^3) \\ (x^4 + 2x^2y^2 - 4x^2 - y^2 + y^4) \\ g = \mathrm{diff}(f, x) \end{aligned}$$

System W_2:

$$\begin{aligned} f = y^4 - 6y^2x + x^2 - 4x^2y^2 + 24x^3 \\ g = \mathrm{diff}(f, x) \end{aligned}$$

System W_3:

$$\begin{aligned} f = ((x-1)^2 + y^2 - 2)((x+1)^2 + y^2 - 2) \\ ((x-1)^2 + (y+2)^2 - 2)((x+1)^2 + (y+2)^2 - 2) \\ g = \mathrm{diff}(f, x) \end{aligned}$$

System W_4:

$$\begin{aligned} f = (x^2 - 2x - 1 + y^2)(x^2 + 2x - 1 + y^2) \\ (x^2 - 2x + 3 + y^2 + 4y) \\ (100000x^2 + 200000x + 299999 + 100000y^2 + 400000y) \\ g = \mathrm{diff}(f, x) \end{aligned}$$

System W_5:

$$\begin{aligned} f = (x^4 + 4x^3 + 6x^2 + 4x + y^4 + 4y^3 + 6y^2 + 4y) \\ (x^4 + 4x^3 + 6x^2 + 4x + y^4 - 4y^3 + 6y^2 - 4y) \\ (x^4 - 4x^3 + 6x^2 - 4x + y^4 + 4y^3 + 6y^2 + 4y) \\ (100000x^4 - 400000x^3 + 600000x^2 - 400000x \\ -1 + 100000y^4 - 400000y^3 + 600000y^2 - 400000y) \\ g = \mathrm{diff}(f, x) \end{aligned}$$

B Sample Usage

For a more up-to-date coverage of the capabilities of the SLV library the reader is urged to visit `http://www.di.uoa.gr/~erga/soft/SLV_index.html` which is the official homepage of the library. SLV library requires a definition for variable `LIBPATH` which should point on the appropriate path where the source code is stored in your system. On the following, we assume that SLV is located under `/opt/AlgebraicLibs/SLV/`. The following is an example for univariate solving:

```
> LIBPATH := "/opt/AlgebraicLibs/SLV/":
> read cat ( LIBPATH, "system.mpl" ):
> f := 3*x^3 - x^2 - 6*x + 2:
> sols := SLV:-solveUnivariate( f ):
> SLV:-display_1 ( sols );
  < x^2-2, [ -93/64, -45/32], -1.414213568 >
  < 3*x-1, [ 1/3, 1/3], 1/3 >
  < x^2-2, [ 45/32, 93/64], 1.414213568 >
```

Note, that the multiplicities of the roots do not appear, although they have been computed. Instead, the third argument of each component in the *printed* list is an approximation of the root. However, whenever possible we provide rational representation of the root.

The following is an example for bivariate solving, where the second root lies in $\mathbb{Z}^2$:

```
> LIBPATH := "/opt/AlgebraicLibs/SLV/":
> read cat ( LIBPATH, "system.mpl" ):
> f := 1+2*x+x^2*y-5*x*y+x^2:
> g := 2*x+y-3:
> bivsols := SLV:-solveGRID ( f, g ):
> SLV:-display_2 ( bivsols );
  < 2*x^2-12*x+1, [ 3, 7], 5.915475965 > ,
  < x^2+6*x-25, [ -2263/256, -35/4], -8.830718995 >

  < x-1, [ 1, 1], 1 > , < x-1, [ 1, 1], 1 >

  < 2*x^2-12*x+1, [ 3/64, 3/32], .8452400565e-1 > ,
  < x^2+6*x-25, [ 23179/8192, 2899/1024], 2.830943108 >
```

Again, just like in the case of univariate solving, the third argument that is printed on the component that describes each algebraic number is an approximation of the number and not the multiplicity of the root. Similarly, one could have used one of the other solvers on the above example by referring to their names, i.e. call the solvers with one of the following commands:

```
> bivsols := SLV:-solveMRUR ( f, g ):
> bivsols := SLV:-solveGRUR ( f, g ):
```

For those interested in the numerical values or rough approximations of the solutions one can get the appropriate output via `display_float_1` and `display_float_2` procedures. Hence, for the above examples we have:

```
> SLV:-display_float_1 ( sols );
  < -1.4142136 >
```

```
  <  0.3333333 >
  <  1.4142136 >
> SLV:-display_float_2 ( bivsols );
  [   5.9154759, -8.8309519, ]
  [   1.0000000,  1.0000000, ]
  [   0.0845241,  2.8309519, ]
```

Consider the list `sols` of $\mathbb{R}_{alg}$ numbers that was returned in the univariate case above; the following are examples on the usage of the `signAt` function provided by our Filtered Kernel[4]:

```
> FK:-signAt( 2*x + 3, sols[1] );
                                1
> FK:-signAt( x^2*y + 2, sols[3], sols[1] );
                                -1
```

Our class on Polynomial Remainder Sequences[5] exports functions allowing the computation of Subresultant and Sturm-Habicht sequences. Let $f, g \in \mathbb{Z}[x, y]$, then you can use *any* of the following commands in order to compute the desired PRS:

```
L := PRS:-StHa ( f, g, y ):
L := PRS:-StHaByDet ( f, g, y ):
L := PRS:-subresPRS ( f, g, y ):
L := PRS:-SubResByDet ( f, g, y ):
```

`PrintPRS` is used for viewing the PRS. For example, let f, g be those from the example on Bivariate Solving above:

```
> L := PRS:-subresPRS ( f, g, y ):
> PRS:-PrintPRS( L );
                  / 2      \                 2
                  \x  - 5 x/ y + 1 + 2 x + x
                         y + 2 x - 3
                      3       2
                    2 x  - 14 x  + 13 x - 1
```

Finally, the variance of the above sequence evaluated at $(1, 0)$ can be computed by:

```
> G := PRS:-Eval ( L, 1, 0 );
                          G := [4, -1, 0]
> PRS:-var( G );
                                1
```

[4]Located in file: `FK.mpl`
[5]Located in file: `PRS.mpl`

References

[Abb06] J. Abbott. Quadratic interval refinement for real roots. In *ISSAC 2006, poster presentation*, 2006. http://www.dima.unige.it/ abbott/.

[AM88] D. Arnon and S. McCallum. A polynomial time algorithm for the topological type of a real algebraic curve. *JSC*, 5:213–236, 1988.

[BK86] E. Brieskorn and H. Knörrer. *Plane Algebraic Curves*. Birkhäuser, Basel, 1986.

[BPM06] 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.

[Can88] J. Canny. Some algebraic and geometric computations in PSPACE. In *Proc. STOC*, pages 460–467, 1988.

[CFPR06] Frédéric Cazals, Jean-Charles Faugère, Marc Pouget, and Fabrice Rouillier. The implicit structure of ridges of a smooth parametric surface. *Comput. Aided Geom. Des.*, 23(7):582–598, 2006.

[DET07] D. I. Diochnos, I. Z. Emiris, and E. P. Tsigaridas. On the complexity of real solving bivariate systems. Research Report 6116, INRIA, 02 2007. https://hal.inria.fr/inria-00129309.

[DSY05] 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 81–93, Beijing, China, 2005.

[EMT07] 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*, LNCS (to appear). Springer Verlag, 2007. also available in www.inria.fr/rrrt/rr-5897.html.

[ESY06] A. Eigenwillig, V. Sharma, and C. K. Yap. Almost tight recursion tree bounds for the descartes method. In *ISSAC*, pages 71–78, New York, NY, USA, 2006. ACM Press.

[ET05] 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.

[GVEK96] 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.

[GVLRR89] L. González-Vega, H. Lombardi, T. Recio, and M-F. Roy. Sturm-Habicht Sequence. In *ISSAC*, pages 136–146, 1989.

[GVN02] L. Gonzalez-Vega and I. Necula. Efficient topology determination of implicitly defined algebraic plane curves. *Computer Aided Geometric Design*, 19(9):719–743, December 2002.

[Ker06] M. Kerber. Analysis of real algebraic plane curves. Diploma thesis, MPI Saarbrücken, 2006.

[Klo95] J. Klose. Binary segmentation for multivariate polynomials. *J. Complexity*, 11(3):330–343, 1995.

[KSP05] K.H. Ko, T. Sakkalis, and N.M. Patrikalakis. Resolution of multiple roots of nonlinear polynomial systems. *Int. J. of Shape Modeling*, 11(1):121–147, 2005.

[LR01] T. Lickteig and M-F. Roy. Sylvester-Habicht Sequences and Fast Cauchy Index Computation. *JSC*, 31(3):315–341, 2001.

[LRSED00] H. Lombardi, M-F. Roy, and M. Safey El Din. New Structure Theorem for Subresultants. *JSC*, 29(4-5):663–689, 2000.

[Mil92] 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.

[MP05] B. Mourrain and J-P. Pavone. Subdivision methods for solving polynomial equations. Technical Report RR-5658, INRIA Sophia-Antipolis, 2005.

[MPS+06] Bernard Mourrain, Sylvain Pion, Susanne Schmitt, Jean-Pierre Técourt, Elias Tsigaridas, and Nicola Wolpert. Algebraic issues in computational geometry. In Jean-Daniel Boissonnat and Monique Teillaud, editors, *Effective Computational Geometry for Curves and Surfaces*, pages 117–155. Springer-Verlag, 2006.

[MS99] M. Mignotte and D. Stefanescu. *Polynomials: An algorithmic approach.* Springer, 1999.

[MT00] B. Mourrain and P. Trébuchet. Solving projective complete intersection faster. In C. Traverso, editor, *Proc. Intern. Symp. on Symbolic and Algebraic Computation*, pages 231–238. New-York, ACM Press., 2000.

[Neu90] Arnold Neumaier. *Interval Methods for Systems of Equations.* Cambridge University Press, 1990.

[Pan02] V.Y. Pan. Univariate polynomials: Nearly optimal algorithms for numerical factorization and rootfinding. *JSC*, 33(5):701–733, 2002.

[PRS93] 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.

[Rei97] D. Reischert. Asymptotically fast computation of subresultants. In *ISSAC*, pages 233–240, 1997.

[Ren89] J. Renegar. On the worst-case arithmetic complexity of approximating zeros of systems of polynomials. *SIAM J. Computing*, 18:350–370, 1989.

[Rou99] F. Rouillier. Solving zero-dimensional systems through the rational univariate representation. *J. of AAECC*, 9(5):433–461, 1999.

[Sak89] T. Sakkalis. Signs of algebraic numbers. *Computers and Mathematics*, pages 131–134, 1989.

[SF90] T. Sakkalis and R. Farouki. Singular points of algebraic curves. *JSC*, 9(4):405–421, 1990.

[vHM02] M. van Hoeij and M. Monagan. A modular GCD algorithm over number fields presented with multiple extensions. In *ISSAC*, pages 109–116, July 2002.

[vzGL03] J. von zur Gathen and T. Lücking. Subresultants revisited. *TCS*, 1-3(297):199–239, 2003.

[Wol02] N. Wolpert. *An Exact and Efficient Approach for Computing a Cell in an Arrangement of Quadrics*. PhD thesis, MPI fuer Informatik, October 2002.

[WS05] N. Wolpert and R. Seidel. On the Exact Computation of the Topology of Real Algebraic Curves. In *SoCG*, pages 107–115. ACM, 2005.

[Yap00] C.K. Yap. *Fundamental Problems of Algorithmic Algebra*. Oxford University Press, New York, 2000.

Unité de recherche INRIA Lorraine
LORIA, Technopôle de Nancy-Brabois - Campus scientifique
615, rue du Jardin Botanique - BP 101 - 54602 Villers-lès-Nancy Cedex (France)

Unité de recherche INRIA Futurs : Parc Club Orsay Université - ZAC des Vignes
4, rue Jacques Monod - 91893 ORSAY Cedex (France)
Unité de recherche INRIA Rennes : IRISA, Campus universitaire de Beaulieu - 35042 Rennes Cedex (France)
Unité de recherche INRIA Rhône-Alpes : 655, avenue de l'Europe - 38334 Montbonnot Saint-Ismier (France)
Unité de recherche INRIA Rocquencourt : Domaine de Voluceau - Rocquencourt - BP 105 - 78153 Le Chesnay Cedex (France)
Unité de recherche INRIA Sophia Antipolis : 2004, route des Lucioles - BP 93 - 06902 Sophia Antipolis Cedex (France)

Éditeur
INRIA - Domaine de Voluceau - Rocquencourt, BP 105 - 78153 Le Chesnay Cedex (France)
http://www.inria.fr
ISSN 0249-6399