An Elimination Method for Solving Bivariate Polynomial Systems: Eliminating the Usual Drawbacks

Eric Berberich* Pavel Emeliyanenko* Michael Sagraloff*

Abstract

We present an exact and complete algorithm to isolate the real solutions of a zero-dimensional bivariate polynomial system. The proposed algorithm constitutes an elimination method which improves upon existing approaches in a number of points. First, the amount of purely symbolic operations is significantly reduced, that is, only resultant computation and square-free factorization is still needed. Second, our algorithm neither assumes generic position of the input system nor demands for any change of the coordinate system. The latter is due to a novel inclusion predicate to certify that a certain region is isolating for a solution. Our implementation exploits graphics hardware to expedite the resultant computation. Furthermore, we integrate a number of filtering techniques to improve the overall performance. Efficiency of the proposed method is proven by a comparison of our implementation with two state-of-the-art implementations, that is, Lgp and Maple's Isolate. For a series of challenging benchmark instances, experiments show that our implementation outperforms both contestants.

1 Introduction

Finding the real solutions of a bivariate polynomial system is a fundamental problem with numerous applications in computational geometry, computer graphics and computer aided geometric design. In particular, topology and arrangement computations for algebraic curves [7, 13, 14, 20, 21] crucially rely on the computation of common intersection points of the given curves (and also the curves defined by their partial derivatives). For the design of robust and certified algorithms, we aim for exact methods to determine isolating regions for all solutions. Such methods should be capable of handling any input, that is, even systems with multiple solutions. The proposed algorithm Bisolve constitutes such an exact and complete approach. Its input is a *zero-dimensional* (i.e., there exist only finitely many solutions) polynomial system $f(x,y) = g(x,y) = 0$ defined by two bivariate polynomials with integer coefficients. Bisolve computes disjoint boxes $B_1,\dots,B_m \subset \mathbb{R}^2$ for *all real solutions*, where each box B_i contains exactly one solution (i.e., B_i is isolating). In addition, the boxes can be refined to an arbitrary small size.

Main results. Bisolve is a classical elimination method which follows the same basic idea as the GRID method from [12] or the INSULATE method from [36] for computing the topology of an algebraic planar curve.[1] That is, they all consider several projection directions to derive a set of candidates of possible solutions and eventually identify those candidates which are actually solutions.

More precisely, in a first step, we separately eliminate the variables x and y by means of a resultant computation. Then, in the second step, for each possible candidate (represented as pair of projected solutions in x- and y-direction), we check whether it actually constitutes a solution of the given system or not. The proposed method comes with a number of improvements compared to the aforementioned approaches and also to other existing elimination techniques [3, 13, 25, 29, 30]. First, we tremendously reduced the amount of purely symbolic computations, namely, our method only demands for resultant and gcd computation of univariate polynomials with integer coefficients. Second, our implementation profits from a novel approach [16, 17] to compute resultants exploiting the power of Graphics Processing Unite (GPUs). We remark that, in comparison to the classical resultant computation on the CPU, the GPU implementation is typically more than 100-times faster. Our experiments show that, for the considered instances, the resultant computation is no longer a "global" bottleneck of an elimination approach. Third, the proposed method never uses any kind of a coordinate transformation, even for non-generic input.[2] The

*Max-Planck-Institut für Informatik, Saarbrücken, Germany, {eric,asm,msagralo}@mpi-inf.mpg.de

[1]For the analysis of a planar curve $C = \{(x,y) \in \mathbb{R}^2 : f(x,y) = 0\}$, it is crucial to find the solutions of $f = f_y = 0$. The method in [36] uses several projection directions to find these solutions.

[2]The system $f = g = 0$ is non-generic if there exist two solutions sharing a common coordinate.

latter is due to a novel inclusion predicate which combines information from the resultant computation and a homotopy argument to prove that a certain candidate box is isolating for a solution. Since we never apply a change of coordinates, our method particularly profits in the case where f and g are sparse or where we are only interested in "local" solutions within a given box. Finally, we integrated a series of additional filtering techniques which allow us to significantly speed up the computation for many instances.

We implemented our algorithm as a prototypical package of CGAL [40] and ran our software on numerous challenging benchmark instances. For comparison, we considered two currently state-of-the-art implementations, that is, ISOLATE (based on RS by Fabrice Rouillier with ideas from [29]) and LGP by Xiao-Shan Gao et al. [8], both interface in Maple 13. Our experiments show that our method is efficient as it outperforms both contestants for most instances. More precisely, our method is comparable for all considered instances and typically between 5 and 10-times faster. For some instances, we even improve by a factor of 50 and more. Our filters apply to many input systems and crucially contribute to the overall performance. We further remark that the gain in performance is not solely due to the resultant computation on the GPU but rather due to the combination of the sparse use of purely symbolic computations and efficient (approximate) subroutines. We prove the latter fact by providing running times with and without fast GPU-resultant computation.

Related Work. Since polynomial root solving is such an important problem in several fields, plenty of distinct approaches exist and many textbooks are dedicated to this subject. We mainly distinguish between two kinds of methods.

The first comprises non-certified or non-complete methods which give, in contrast to our goal here, no guarantee on correctness or termination (e.g., if multiple roots exists). Representatives of this category are numerical (e.g., homotopy methods [37]) or subdivision methods[3] (e.g., [2, 6, 27]). A major strength of these methods is that they are very efficient for most instances due to their use of approximate computations such as provided by IntBis, ALIAS, IntLab or MPFI.

The second category consists of certified and complete methods to which ours is to be added. So far, only elimination methods based on *(sparse) resultants*, *rational univariate representation*, *Groebner bases* or *eigenvalues* have proven to be reasonably efficient representatives of this category; see, for instance, [11, 28, 39, 41] for introductions to such symbolic approaches. Common to all these methods is that they combine a projection and a lifting step similar to the proposed approach. Recent exact and complete implementations for computing the topology of algebraic curves and surfaces [5, 7, 14, 20, 21, 36] also make use of such elimination techniques. However, already this low dimensional application shows the main drawback of elimination methods, that is, they tremendously suffer from costly symbolic computations. Furthermore, the given system might be in non-generic position which makes the lifting step non-trivial. In such "hard situations", the existing approaches perform a coordinate transformation (or project in generic direction) which eventually increases the complexity of the input polynomials. In particular, if we are only interested in "local" solutions within a given box, such methods induce a huge overhead of purely symbolic computations. The proposed algorithm constitutes a contribution in two respects: The number of symbolic steps are crucially reduced and partially (resultant computation) outsourced to the GPU. In addition, generic and non-generic situations are treated in the same manner and, thus, a coordinate transformation which induces an overhead of symbolic computations is no longer needed.

[3]Subdivision methods can be made certifying and complete when considering worst case separation bounds for the solutions, an approach which has not shown effective in practice so far.

2 Setting

The *input* of our algorithm is the following polynomial system

$$(2.1)\qquad \begin{cases} f(x,y) = \displaystyle\sum_{i,j\in\mathbb{N}:i+j\le m} f_{ij}x^iy^j = 0, \\ g(x,y) = \displaystyle\sum_{i,j\in\mathbb{N}:i+j\le n} g_{ij}x^iy^j = 0, \end{cases}$$

where f, $g \in \mathbb{Z}[x,y]$ are polynomials of total degrees m and n, respectively. We also write

$$f(x,y) = \sum_{i=0}^{m_x} f_i^{(x)}(y)x^i = \sum_{i=0}^{m_y} f_i^{(y)}(x)y^i \quad \text{and}$$

$$g(x,y) = \sum_{i=0}^{n_x} g_i^{(x)}(y)x^i = \sum_{i=0}^{n_y} g_i^{(y)}(x)y^i,$$

where $f_i^{(y)}$, $g_i^{(y)} \in \mathbb{Z}[x]$, $f_i^{(x)}$, $g_i^{(x)} \in \mathbb{Z}[y]$ and m_x, n_x and m_y, n_y denote the degrees of f and g considered as polynomials in x and y, respectively. Throughout the paper, it is assumed that f and g have no common factors.[4] Hence, the set $V_{\mathbb{C}} := \{(x,y) \in \mathbb{C}^2 | f(x,y) =$

[4]Otherwise, f and g have to be decomposed into common and non-common factors (not part of our algorithm).

$g(x,y) = 0\}$ of (complex) solutions of (2.1) is zero-dimensional and consists, by Bézout's theorem, of at most $m \cdot n$ distinct elements.

Our algorithm *outputs* disjoint boxes $B_k \subset \mathbb{R}^2$ such that the union of all B_k contains all *real* solutions

$$V_{\mathbb{R}} := \{(x,y) \in \mathbb{R}^2 | f(x,y) = g(x,y) = 0\} = V_{\mathbb{C}} \cap \mathbb{R}^2$$

of (2.1) and each B_k is *isolating*, that is, it contains exactly one solution.

Notation. For an interval $I = (a,b) \subset \mathbb{R}$, $m_I := (a+b)/2$ denotes the *center* and $r_I := (b-a)/2$ the *radius* of I. For an arbitrary $m \in \mathbb{C}$ and $r \in \mathbb{R}^+$, $\Delta_r(m)$ denotes the disc with center m and radius r.

3 The Algorithm

3.1 Resultants Our method is based on well known elimination techniques. We consider the projections

$$\begin{aligned} V_{\mathbb{C}}^{(x)} &:= \{x \in \mathbb{C} | \exists y \in \mathbb{C} \text{ with } f(x,y) = g(x,y) = 0\}, \\ V_{\mathbb{C}}^{(y)} &:= \{y \in \mathbb{C} | \exists x \in \mathbb{C} \text{ with } f(x,y) = g(x,y) = 0\} \end{aligned}$$

of all complex solutions $V_{\mathbb{C}}$ onto the x- and y-coordinate. Resultant computation is a well studied tool to obtain an algebraic description of these projection sets, that is, polynomials whose roots are exactly the projections of the solution set $V_{\mathbb{C}}$. The resultant $R^{(y)} = \operatorname{res}(f,g,y)$ of f and g with respect to the variable y is the determinant of the $(m_y + n_y) \times (m_y + n_y)$ *Sylvester matrix*:

$$S^{(y)}(f,g) := \begin{bmatrix} f_{m_y}^{(y)} & f_{m_y-1}^{(y)} & \dots & f_0^{(y)} & 0 \dots & 0 \\ \vdots & \ddots & \ddots & & \ddots & \vdots \\ 0 & \dots\ 0 & f_{m_y}^{(y)} & f_{m_y-1}^{(y)} & \dots & f_0^{(y)} \\ g_{n_y}^{(y)} & g_{n_y-1}^{(y)} & \dots & g_0^{(y)} & 0 \dots & 0 \\ \vdots & \ddots & \ddots & & \ddots & \vdots \\ 0 & \dots\ 0 & g_{n_y}^{(y)} & g_{n_y-1}^{(y)} & \dots & g_0^{(y)} \end{bmatrix}$$

From the definition, it follows that $R^{(y)} \in \mathbb{Z}[x]$ has degree less than or equal to $m \cdot n$. The resultant $R^{(x)} = \operatorname{res}(f,g,x)$ of f and g with respect to x is defined in completely analogous manner by considering f and g as polynomials in x instead of y. As mentioned above the resultant polynomials have the following important property (cf. [4] for a proof):

THEOREM 3.1. *The roots of $R^{(y)}$ are exactly the projections of the solutions of (2.1) onto the x-coordinate and the roots of the greatest common divisor $h^{(y)}(x) := \gcd(f_{m_y}(x), g_{n_y}(x))$ of the leading coefficients of f and g. More precisely,*

$$\{x \in \mathbb{C} | R^{(y)}(x) = 0\} = V_{\mathbb{C}}^{(x)} \cup \{x \in \mathbb{C} | h^{(y)}(x) = 0\}$$

For $R^{(y)}$, a corresponding result holds:

$$\{y \in \mathbb{C} | R^{(x)}(y) = 0\} = V_{\mathbb{C}}^{(y)} \cup \{y \in \mathbb{C} | h^{(x)}(y) = 0\},$$

where $h^{(x)}(y) := \gcd(f_{m_x}(y), g_{n_x}(y))$. The multiplicity of a root α of $R^{(y)}$ ($R^{(x)}$) is the sum[5] *of the intersection multiplicities*[6] *of all solutions of (2.1) with x-coordinate (y-coordinate) α.*

3.2 Isolating the Solutions: Project, Separate and Validate We start with the following high level description of the proposed algorithm which decomposes into three subroutines: In the first step (PROJECT), we project the complex solutions $V_{\mathbb{C}}$ of (2.1) onto the x- and onto the y-axis. More precisely, we compute the restrictions $V_{\mathbb{R}}^{(x)} := V_{\mathbb{C}}^{(x)} \cap \mathbb{R}$ and $V_{\mathbb{R}}^{(y)} := V_{\mathbb{C}}^{(y)} \cap \mathbb{R}$. of the complex projection sets $V_{\mathbb{C}}^{(x)}$ and $V_{\mathbb{C}}^{(y)}$ to the real axes and isolating intervals for their elements. Obviously, the real solutions $V_{\mathbb{R}}$ are contained in the cross product $\mathcal{C} := V_{\mathbb{R}}^{(x)} \times V_{\mathbb{R}}^{(y)} \subset \mathbb{R}^2$. In the second step (SEPARATE), we compute isolating discs which well separate the projected solutions from each other. The latter prepares the third step (VALIDATE) in which candidates of $\mathcal{C}$ are either discarded or certified to be a solution of (2.1). Our *main theoretical contribution* is the introduction of a novel predicate to ensure that a certain candidate $(\alpha,\beta) \in \mathcal{C} \cap V_{\mathbb{R}}$ actually fulfills $f(\alpha,\beta) = g(\alpha,\beta) = 0$ (cf. Theorem 3.4). For all candidates $(\alpha,\beta) \in \mathcal{C} \backslash V_{\mathbb{R}}$, simple interval arithmetic suffices to exclude (α,β) as a solution of (2.1).

We remark that, in order to increase the efficiency of our implementation, we also introduce additional filtering techniques to eliminate many of the candidates in $\mathcal{C}$. However, for the sake of clarity, we refrain from integrating our filtering techniques in the following description of the three subroutines. Filtering techniques are covered separately in Section 4.2. Section 4.1 briefly discusses a highly parallel algorithm on the graphics hardware to accelerate computations of the resultants needed in the first step.

PROJECT: We compute the resultant $R := R^{(y)} = \operatorname{res}(f,g,y) \in \mathbb{Z}[x]$ and a square-free factorization of R. More precisely, we determine square-free and pairwise coprime factors $r_i \in \mathbb{Z}[x]$, $i = 1, \dots, \deg(R)$, such that $R(x) = \prod_{i=1}^{\deg(R)} (r_i(x))^i$. We remark that, for some

[5]For a root α of $h^{(y)}(x)$ (or $h^{(x)}(y)$), the intersection multiplicity of f and g at the "infinite point" (α,∞) (or (∞,α)) has also been taken into account. For simplicity, we decided not to consider the more general projective setting.

[6]The multiplicity of a solution (x_0,y_0) of (2.1) is defined as the dimension of the localization of $\mathbb{C}[x,y]/(f,g)$ at (x_0,y_0) considered as $\mathbb{C}$-vector space (cf. [4, p.148])

$i \in \{1,\dots,\deg(R)\}$, $r_i(x) = 1$. Yun's algorithm [18, Alg. 14.21] constructs such a square-free factorization by essentially computing greatest common divisors of R and its higher derivatives in an iterative way. Next, we isolate the real roots $\alpha_{i,j}$, $j = 1,\dots,\ell_i$, of the polynomials r_i. That is, we determine disjoint isolating intervals $I(\alpha_{i,j}) \subset \mathbb{R}$ such that each interval $I(\alpha_{i,j})$ contains exactly one root (namely, $\alpha_{i,j}$) of r_i and the union of all $I(\alpha_{i,j})$, $j = 1,\dots,\ell_i$, covers all real roots of r_i. For the real root isolation, we consider the Descartes method [10, 31] as a suited algorithm. From the square-free factorization we know that $\alpha_{i,j}, j = 1,\dots,\ell_i$, is a root of R with multiplicity i.

SEPARATE: We separate the real roots of $R = R^{(y)}$ from all other (complex) roots of R, a step which is crucial for the final validation. More precisely, let $\alpha = \alpha_{i_0,j_0}$ be the j_0-th real root of the polynomial r_{i_0}, where $i_0 \in \{1,\dots,\deg(R)\}$ and $j_0 \in \{1,\dots,\ell_{i_0}\}$ are arbitrary indices. We refine the corresponding isolating interval $I = (a,b) := I(\alpha)$ such that the disc $\Delta_{8r_I}(m_I)$ does not contain any root of $R^{(y)}$ except α. For the refinement of I, we use quadratic interval refinement (QIR for short) [1, 24] which constitutes a highly efficient method because of its simple tests and the fact that it eventually achieves quadratic convergence.

In order to test whether the disc $\Delta_{8r_I}(m_I)$ isolates α from all other roots of R, we consider an approach which was first introduced in [35]. It is based on the following test:

$$T_K^p(m,r) : |p(m)| - K \sum_{k\geq 1} \left|\frac{p^{(k)}(m)}{k!}\right| r^k > 0,$$

where $p \in \mathbb{R}[x]$ denotes an arbitrary polynomial and m, r, K arbitrary real values. Then, the following theorem holds:[7]

THEOREM 3.2. *Consider a disk $\Delta = \Delta_m(r) \subset \mathbb{C}$ with center m and radius r.*

1. *If $T_K^p(m,r)$ holds for some $K \geq 1$, then the closure $\overline{\Delta}$ of Δ contains no root of p.*
2. *If $T_K^{p'}(m,r)$ holds for a $K \geq \sqrt{2}$, then $\overline{\Delta}$ contains at most one root of p.*

Proof. (1) follows from a straightforward computation: For each $z \in \overline{\Delta}$, we have

$$p(z) = p(m + (z-m)) = p(m) + \sum_{k\geq 1} \frac{p^{(k)}(m)}{k!}(z-m)^k$$

and, thus,

$$\frac{|p(z)|}{|p(m)|} \geq 1 - \frac{1}{|p(m)|} \cdot \sum_{k\geq 1} \frac{|p^{(k)}(m)|}{k!}|z-m|^k > \left(1 - \frac{1}{K}\right)$$

since $|z-m| \leq r$ and $T_K^p(m,r)$ holds. In particular, for $K \geq 1$, the above inequality implies $|p(z)| > 0$ and, thus, p has no root in $\overline{\Delta}$.

It remains to show (2): If $T_K^{p'}(m,r)$ holds, then, for any point $z \in \overline{\Delta}$, the derivative $p'(z)$ differs from $p'(m)$ by a complex number of absolute value less than $|p'(m)|/K$. Consider the triangle spanned by the points 0, $p'(m)$ and $p'(z)$, and let α and β denote the angles at the points 0 and $p'(z)$, respectively. From the Sine Theorem, it follows that

$$|\sin\alpha| = |p'(m) - p'(z)| \cdot \frac{|\sin\gamma|}{|p'(m)|} < \frac{1}{K}.$$

Thus, the arguments of $p'(m)$ and $p'(z)$ differ by less than $\arcsin(1/K)$ which is smaller than or equal to $\pi/4$ for $K \geq \sqrt{2}$. Assume that there exist two roots $a, b \in \overline{\Delta}$ of p. Since $a = b$ implies $p'(a) = 0$, which is not possible as $T_1^{p'}(m,r)$ holds, we can assume that $a \neq b$. We split f into its real and imaginary part, that is, we consider $p(x+iy) = u(x,y) + iv(x,y)$ where $u, v : \mathbb{R}^2 \to \mathbb{R}$ are two bivariate polynomials. Then, $p(a) = p(b) = 0$ and so $u(a) = v(a) = u(b) = v(b) = 0$. But $u(a) = u(b) = 0$ implies, due to the Mean Value Theorem in several real variables, that there exists a $\phi \in [a,b]$ such that

$$\nabla u(\phi) \perp (b-a).$$

Similarly, $v(a) = v(b) = 0$ implies that there exists a $\xi \in [a,b]$ such that $\nabla v(\xi) \perp (b-a)$. But $\nabla v(\xi) = (v_x(\xi), v_y(\xi)) = (-u_y(\xi), u_x(\xi))$, thus, it follows that $\nabla u(\xi) \parallel (b-a)$. Therefore, $\nabla u(\psi)$ and $\nabla u(\xi)$ must be perpendicular. Since $p' = u_x + iv_x = u_x - iu_y$, the arguments of $p'(\psi)$ and $p'(\xi)$ must differ by $\pi/2$. This contradicts our above result that both differ from the argument of $p'(m)$ by less than $\pi/4$, thus, (2) follows.□

Theorem 3.2 now directly applies to the above scenario, where $p = r_{i_0}$ and $r = 8r_I$. More precisely, I is refined until $T_{3/2}^{(r_{i_0})'}(m_I, 8r_I)$ and $T_1^{r_i}(m_I, 8r_I)$ holds for all $i \neq i_0$. If the latter two conditions are fulfilled, $\Delta_{8r_I}(m_I)$ isolates α from all other roots of R. In this situation, we obtain a lower bound $LB(\alpha)$ for $|R(z)|$ on the boundary of $\Delta(\alpha) := \Delta_{2r_I}(m_I)$:

LEMMA 3.1. *Let I be an interval which contains a root α of r_{i_0}. If $T_{3/2}^{(r_{i_0})'}(m_I, 8r_I)$ and $T_1^{r_i}(m_I, 8r_I)$ holds for all $i \neq i_0$, then the disc $\Delta(\alpha) = \Delta_{2r_I}(m_I)$ isolates α*

[7] For a similar result, the reader may also consider [34] where a corresponding test is introduced which is based on interval arithmetic only.

from all other (complex) roots of R and, for any z on the boundary $\partial\Delta(\alpha)$ of $\Delta(\alpha)$, it holds that

$$|R(z)| > LB(\alpha) := 2^{-i_0-\deg(R)}|R(m_I - 2r_I)|.$$

Proof. $\Delta(\alpha)$ is isolating as already $\Delta_{8r_I}(m_I)$ is isolating. Then, let $\beta \neq \alpha$ be an arbitrary root of R and $d := |\beta - m_I| > 8r_I$ the distance between β and m_I. Then, for any point $z \in \partial\Delta(\alpha)$, it holds that

$$\frac{|z-\beta|}{|(m_I - 2r_I) - \beta|} > \frac{d-2r_I}{d+2r_I} = 1 - \frac{4r_I}{d+2r_I} > \frac{1}{2}$$

and

$$\frac{|z-\alpha|}{|(m_I - 2r_I) - \alpha|} > \frac{r_I}{3r_I} > \frac{1}{4}.$$

Hence, it follows that

$$\begin{aligned} \frac{|R(z)|}{|R(m_I-2r_I)|} &> \left(\frac{|z-\alpha|}{|(m_I-2r_I)-\alpha|}\right)^{i_0} \cdot \\ &\quad \prod_{\beta\neq\alpha:\ R(\beta)=0} \frac{|z-\beta|}{|(m_I-2r_I)-\beta|} \\ &> 4^{-i_0}2^{-\deg(R)+i_0}, \end{aligned}$$

where each root β occurs as many times in the above product as its multiplicity as a root of R. □

We compute $LB(\alpha) = 2^{-i_0-\deg(R)}|R(m_I - 2r_I)|$ and store the interval $I(\alpha)$, the disc $\Delta(\alpha)$ and the lower bound $LB(\alpha)$ for $|R(z)|$ on the boundary $\partial\Delta(\alpha)$ of $\Delta(\alpha)$.

Proceeding in exactly the same manner for each real root α of $R^{(y)}$, we get an isolating interval $I(\alpha)$, an isolating disc $\Delta(\alpha) = \Delta_{2r_I}(m_I)$ and a lower bound $LB(\alpha)$ for $|R^{(y)}|$ on $\partial\Delta(\alpha)$. For the resultant polynomial $R^{(x)}$, Project and Separate are processed in exactly the same manner: We compute $R^{(x)}$ and a corresponding square-free factorization. Then, for each real root β of $R^{(x)}$, we compute a corresponding isolating interval $I(\beta)$, a disc $\Delta(\beta)$ and a lower bound $LB(\beta)$ for $|R^{(x)}|$ on $\partial\Delta(\beta)$.

Validate: We start with the following theorem:

Theorem 3.3. *Let α and β be arbitrary real roots of $R^{(y)}$ and $R^{(x)}$, respectively. Then,*

1. *the polydisc $\Delta(\alpha,\beta) := \Delta(\alpha)\times\Delta(\beta) \subset \mathbb{C}^2$ contains at most one (complex) solution of (2.1). If $\Delta(\alpha,\beta)$ contains a solution of (2.1), then this solution is real valued and equals (α,β).*
2. *For an arbitrary point $(z_1,z_2) \in \mathbb{C}^2$ on the boundary of $\Delta(\alpha,\beta)$, it holds that*

$$|R^{(y)}(z_1)| > LB(\alpha) \text{ if } z_1 \in \partial\Delta(\alpha), \text{ and}$$
$$|R^{(x)}(z_2)| > LB(\beta) \text{ if } z_2 \in \partial\Delta(\beta).$$

Proof. (1) is an easy consequence from the construction of the discs $\Delta(\alpha)$ and $\Delta(\beta)$. Namely, if $\Delta(\alpha,\beta)$ contains two distinct solutions of (2.1), then they would differ in at least one coordinate. Thus, one of the discs $\Delta(\alpha)$ or $\Delta(\beta)$ would contain two roots of $R^{(y)}$ or $R^{(x)}$. Since both discs are isolating for a root of the corresponding resultant polynomial, it follows that $\Delta(\alpha,\beta)$ contains at most one solution. In the case, where $\Delta(\alpha,\beta)$ contains a solution of (2.1), this solution must be real since, otherwise, $\Delta(\alpha,\beta)$ would also contain a corresponding complex conjugate solution (f and g have real valued coefficients). (2) follows directly from the definition of $\Delta(\alpha,\beta)$, the definition of $LB(\alpha)$, $LB(\beta)$ and Lemma 3.1. □

We denote $B(\alpha,\beta) = I(\alpha)\times I(\beta) \subset \mathbb{R}^2$ a *candidate box* for a real solution of (2.1), where α and β are real roots of $R^{(y)}$ and $R^{(x)}$, respectively. Due to Theorem 3.3, the corresponding "container polydisc" $\Delta(\alpha,\beta) \subset \mathbb{C}^2$ either contains no solution of (2.1) or (α,β) is the only solution contained in $\Delta(\alpha,\beta)$. Hence, for each candidate pair $(\alpha,\beta) \in \mathcal{C}$, it suffices to show that either (α,β) is no solution of (2.1) or the corresponding polydisc $\Delta(\alpha,\beta)$ contains at least one solution. In the following steps, we fix the polydiscs $\Delta(\alpha,\beta)$ whereas the boxes $B(\alpha,\beta)$ are further refined (by further refining the isolating intervals $I(\alpha)$ and $I(\beta)$). We also introduce exclusion and inclusion predicates such that, for sufficiently small $B(\alpha,\beta)$, either (α,β) can be discarded or certified as a solution of (2.1).

In order to *exclude* a candidate box, we use simple interval arithmetic. More precisely, we evaluate $\Box f(B(\alpha,\beta))$ and $\Box g(B(\alpha,\beta))$, where $\Box f$ and $\Box g$ constitute box functions for f and g, respectively: If either $\Box f(B(\alpha,\beta))$ or $\Box g(B(\alpha,\beta))$ does not contain zero, then (α,β) cannot be a solution of (2.1). Vice versa, if (α,β) is not a solution and $B(\alpha,\beta)$ becomes sufficiently small, then either $0 \notin \Box f(B(\alpha,\beta))$ or $0 \notin \Box g(B(\alpha,\beta))$ and our exclusion predicate applies.

It remains to provide an *inclusion predicate*, that is, a method to ensure that a certain candidate $(\alpha,\beta) \in \mathcal{C}$ is actually a solution of (2.1). We first rewrite the resultant polynomial $R^{(y)}$ as

$$R^{(y)}(x) = u^{(y)}(x,y)\cdot f(x,y) + v^{(y)}(x,y)\cdot g(x,y),$$

where $u^{(y)}$, $v^{(y)} \in \mathbb{Z}[x,y]$, that can be expressed as determinants of "Sylvester-like" matrices:

$$U^{(y)} = \begin{vmatrix} f_{m_y}^{(y)} & f_{m_y-1,y}^{(y)} & \dots & f_0^{(y)} & 0 \dots & y^{n_y-1} \\ \vdots & \ddots & \ddots & & \ddots & \vdots \\ 0 & \dots\ 0 & f_{m_y}^{(y)} & f_{m_y-1}^{(y)} & \dots & 1 \\ g_{n_y}^{(y)} & g_{n_y-1}^{(y)} & \dots & g_0^{(y)} & 0 \dots & 0 \\ \vdots & \ddots & \ddots & & \ddots & \vdots \\ 0 & \dots\ 0 & g_{n_y}^{(y)} & g_{n_y-1}^{(y)} & \dots & 0 \end{vmatrix}$$

$$V^{(y)} = \begin{vmatrix} f_{m_y}^{(y)} & f_{m_y-1}^{(y)} & \dots & f_0^{(y)} & 0 \dots & 0 \\ \vdots & \ddots & \ddots & & \ddots & \vdots \\ 0 & \dots\ 0 & f_{m_y}^{(y)} & f_{m_y-1}^{(y)} & \dots & 0 \\ g_{n_y}^{(y)} & g_{n_y-1}^{(y)} & \dots & g_0^{(y)} & 0 \dots & y^{m_y-1} \\ \vdots & \ddots & \ddots & & \ddots & \vdots \\ 0 & \dots\ 0 & g_{n_y}^{(y)} & g_{n_y-1}^{(y)} & \dots & 1 \end{vmatrix}$$

$U^{(y)}$ and $V^{(y)}$ are obtained from $S^{(y)}(f,g)$ by replacing the last column with vectors $(y^{n_y-1} \dots 1\ 0 \dots 0)^T$ and $(0 \dots 0\ y^{m_y-1} \dots 1)^T$ of appropriate size, respectively [19, p. 287]. Both matrices have size $(n_y + m_y) \times (n_y + m_y)$ and univariate polynomials in x (the first $n_y + m_y - 1$ columns) or powers of y (only the last column) or zeros as entries. We now aim for upper bounds for $|u^{(y)}|$ and $|v^{(y)}|$ on the polydisc $\Delta(\alpha,\beta)$. The polynomials $u^{(y)}$ and $v^{(y)}$ have huge coefficients and their computation, either via a signed remainder sequence or via determinant evaluation, is very costly. Hence, we directly derive such upper bounds from the corresponding matrix representations **without computing** $u^{(y)}$ and $v^{(y)}$: Due to Hadamard's bound, $|u^{(y)}|$ is smaller than the product of the 2-norms of the column vectors of $U^{(y)}$. The absolute value of each of the entries of $U^{(y)}$ can be easily upper bounded by using interval arithmetic on a box in $\mathbb{C}^2$ that contains the polydisc $\Delta(\alpha,\beta)$. Hence, we get an upper bound on the $2-$norm of each column vector and, thus, an upper bound $UB(\alpha,\beta,u^{(y)})$ for $|u^{(y)}|$ on $\Delta(\alpha,\beta)$ by multiplying the bounds for the column vectors. In the same manner, we also derive an upper bound $UB(\alpha,\beta,v^{(y)})$ for $|v^{(y)}|$ on $\Delta(\alpha,\beta)$. With respect to our second projection direction, we write $R^{(x)} = u^{(x)} \cdot f + v^{(x)} \cdot g$ with corresponding polynomials $u^{(x)}$, $v^{(x)} \in \mathbb{Z}[x,y]$. In exactly the same manner as done for $R^{(y)}$, we compute corresponding upper bounds $UB(\alpha,\beta,u^{(x)})$ and $UB(\alpha,\beta,v^{(x)})$ for $|u^{(x)}|$ and $|v^{(x)}|$ on $\Delta(\alpha,\beta)$.

THEOREM 3.4. *If there exists an $(x_0,y_0) \in \Delta(\alpha,\beta)$ with*

$$\begin{aligned} UB(\alpha,\beta,u^{(y)}) \cdot |f(x_0,y_0)| &+ \\ UB(\alpha,\beta,v^{(y)}) \cdot |g(x_0,y_0)| &< LB(\alpha) \end{aligned} \tag{3.2}$$

$$\begin{aligned} UB(\alpha,\beta,u^{(x)}) \cdot |f(x_0,y_0)| &+ \\ UB(\alpha,\beta,v^{(x)}) \cdot |g(x_0,y_0)| &< LB(\beta), \end{aligned} \tag{3.3}$$

then $\Delta(\alpha,\beta)$ contains a solution of (2.1) and, thus, $f(\alpha,\beta) = 0$.

Proof. The proof uses a homotopy argument. Namely, we consider the parameterized system

$$\begin{aligned} f(x,y) - (1-t) \cdot f(x_0,y_0) &= \\ g(x,y) - (1-t) \cdot g(x_0,y_0) &= 0, \end{aligned} \tag{3.4}$$

where t is an arbitrary real value in $[0,1]$. For $t = 1$, (3.4) is equivalent to our initial system (2.1). For $t = 0$, (3.4) has a solution in $\Delta(\alpha,\beta)$, namely, (x_0,y_0). The complex solutions of (3.4) continuously depend on the parameter t. Hence, there exists a "solution path" $\Gamma : [0,1] \mapsto \mathbb{C}^2$ which connects $\Gamma(0) = (x_0,y_0)$ with a solution $\Gamma(1) \in \mathbb{C}^2$ of (2.1). We show that $\Gamma(t)$ does not leave the polydisc $\Delta(\alpha,\beta)$ and, thus, (2.1) has a solution in $\Delta(\alpha,\beta)$: Assume that the path $\Gamma(t)$ leaves the polydisc, then there exists a $t' \in [0,1]$ with $(x',y') = \Gamma(t') \in \partial\Delta(\alpha,\beta)$. We assume that $x' \in \partial\Delta(\alpha)$ (the case $y' \in \partial\Delta(\beta)$ is treated in analogous manner). Since (x',y') is a solution of (3.4) for $t = t'$, we must have $|f(x',y')| \le |f(x_0,y_0)|$ and $|g(x',y')| \le |g(x_0,y_0)|$. Hence, it follows that

$$\begin{aligned} |R^{(y)}(x')| &= |u^{(y)}(x',y')f(x',y') + v^{(y)}(x',y')g(x',y')| \\ &\le |u^{(y)}(x',y')| \cdot |f(x',y')| + \\ &\quad |v^{(y)}(x',y')| \cdot |g(x',y')| \\ &\le UB(\alpha,\beta,u^{(y)}) \cdot |f(x_0,y_0)| + \\ &\quad UB(\alpha,\beta,v^{(y)}) \cdot |g(x_0,y_0)| \\ &< LB(\alpha). \end{aligned}$$

This contradicts the fact that $|R^{(y)}(x')|$ is lower bounded by $LB(\alpha)$. It follows that $\Delta(\alpha,\beta)$ contains a solution of (2.1) and, according to Theorem 3.3, this solution must be (α,β). □

Theorem 3.4 now directly applies as an inclusion predicate. Namely, in each refinement of $B(\alpha,\beta)$, we choose an arbitrary $(x_0,y_0) \in B(\alpha,\beta)$ (e.g., the center $(m_{I(\alpha)}, m_{I(\beta)})$) of the candidate box $B(\alpha,\beta)$) and check whether both inequalities (3.2) and (3.3) are fulfilled. If (α,β) is a solution of (2.1), then both inequalities eventually hold and, thus, we have shown that (α,β) is a solution.

We remark that the upper bounds $UB(\alpha,\beta,u^{(y)})$, $UB(\alpha,\beta,v^{(y)})$, $UB(\alpha,\beta,u^{(x)})$ and $UB(\alpha,\beta,v^{(y)})$ are far from being optimal. Nevertheless, our inclusion predicate is still efficient since we can approximate the potential solution (α,β) with quadratic convergence due to QIR. Hence, the values $f(x_0, y_0)$ and $g(x_0, y_0)$ become very small after a few iterations. In order to improve the above upper bounds, we propose to consider more sophisticated methods from numerical analysis and matrix perturbation theory [22, 32]. Finally, we would like to emphasize that our method applies particularly well to the situation where we are only interested in the solutions of (2.1) within a given box $\mathcal{B} = [A, B] \times [C, D] \subset \mathbb{R}^2$. Though $R^{(y)}$ ($R^{(x)}$) capture all (real and complex) projections of the solutions of the system, we only have to search the real ones contained within the interval $[A, B]$ ($[C, D]$). Then, only candidate boxes within $\mathcal{B}$ have to be considered in Separate and Validate. Since the computation of the resultants is relatively cheap due to our fast implementation on the GPU our method is particularly well suited to search for local solutions.

4 Speedups

4.1 Resultants on graphics hardware Computing the resultants of bivariate polynomials is an important "symbolic part" of our algorithm. Despite a large body of research existing on this subject, symbolic computations still constitute a large bottleneck in many algorithms and substantially limit their range of applicability. We use a novel approach exploiting the power of GPUs to dramatically reduce the time for computing resultants. In this section, we briefly discuss the algorithm; we refer the reader to [17, 16] for details.

Our approach is based on the classical "divide-conquer-combine" modular algorithm by Collins [9]. The algorithm can be summarized in the following steps. 1. Apply modular and evaluation homomorphisms to map the problem to computing a large set of problems over a simple domain. 2. Compute a set of resultants over a prime field. 3. Recover the resultant through polynomial interpolation and Chinese remaindering.

Unfortunately, Collins' algorithm in its original form is not suitable for a realization on the GPU. This is because the amount of parallelism exposed by a modular algorithm is still too low to satisfy the needs of the massively-threaded architecture. To overcome this limitation we reduce the problem to computations with *structured matrices* because matrix operations typically map very well to the GPU's threading model. When the problem is expressed in terms of linear algebra, all data dependencies are usually made explicit (though at the cost of some additional work) allowing for fine-grained parallelism which is a key ingredient in implementing the GPU algorithm.

As a result, all steps of the algorithm except the initial modular reduction and partly the Chinese remaindering are run on the graphics hardware, thereby minimizing the amount of work to be done on the CPU. For expository purposes, we outline here the computation of univariate resultants in more detail.[8]

Suppose, f and g are polynomials in $\mathbb{Z}[x]$ of degrees m and n, respectively. It is clear that the resultant of f and g reduces to the triangular factorization of the Sylvester matrix S (see Section 3.1). The matrix $S \in \mathbb{Z}^{r \times r}$ ($r = m + n$) is structured as it satisfies the displacement equation [23]:

$$S - Z_r S A^T = G B^T,$$

with $A = Z_m \oplus Z_n$ and $G, B \in \mathbb{Z}^{r \times 2}$, where $Z_s \in \mathbb{Z}^{s \times s}$ is a down-shift matrix zeroed everywhere except for 1's on the first subdiagonal. Accordingly, the *generators* G, B are matrices whose entries can be deduced from the matrix S by inspection. Hence, we can apply the generalized Schur algorithm which operates on the matrix generators to compute the matrix factorization in $\mathcal{O}(r^2)$ time; see [23, p. 323].

In short, the Schur algorithm is an iterative procedure: In each step, it brings the matrix generators in a "special form" from which triangular factors can easily be deduced based on the displacement equation. Using division-free modifications this procedure can be performed in a finite field giving rise to the factorization algorithm running in $\mathcal{O}(r)$ time using r processors.

Although, the theoretical background of the algorithm is well-established, it does not say much about the realization. To give the basic idea, observe that, there are two levels of parallelism available on the graphics processor. *Block-level* parallelism allows numerous thread blocks to execute concurrently without an explicit communication (or synchronization) between them. In its turn, threads from the same block are also executed in parallel and can communicate using synchronization barriers and shared memory. The latter one is referred to as *thread-level* parallelism. Suppose that we have applied modular and evaluation homomorphisms to reduce the problem to computing N univariate resultants for each of M moduli. Hence, we can launch a grid of $N \times M$ thread blocks where each individual block computes one univariate resultant using the Schur algorithm above. The pseudocode for (sequential) algorithm can be found in [16, Section 4.2]. Now, exploiting thread-level parallelism, we assign one thread to one row of each of the generator matrices, that is, to four elements (because $G, B \in \mathbb{Z}^{r \times 2}$). In each iteration, one thread updates its associated generator rows (multiplies them by 2×2 transformation matrix). This explains the basic routine of the algorithm.

[8]Remark that, the algorithm described in [17, 16] compute resultants of bivariate polynomials, however dealing with univariate polynomials constitutes a "basic building block" of the algorithm.

It is important to understand that there is a number of factors governing the performance of the GPU algorithm. These, for instance, include: thread occupancy, shared memory usage, register pressure, synchronization overhead, and a concrete realization of the modular arithmetic on the GPU. These topics are covered in the papers cited above.

Once the univariate resultants $z_i^{(p)}$ for each modulus $\mathbb{Z}_p$ and each evaluation point $x_i \in \mathbb{Z}_p$ have been computed, we interpolate the resultant polynomial $R^{(y)}(x)$ in a prime field $\mathbb{Z}_p$ and eventually lift it via Chinese remaindering to obtain an integer solution. These steps are also executed (partly) on the GPU. We remark that polynomial interpolation corresponds to solving a Vandermonde system.[9] Again, exploiting the structure of Vandermonde matrix we can use the Schur algorithm to solve the system in a small parallel time.

4.2 Filters Besides the parallel resultant computation, our algorithm elaborates a number of filtering techniques to early validate a majority of the candidates.

As first step, we group candidates along the same vertical line (a *fiber*) at an x-coordinate α (a root of $R^{(y)}$) to process them together. This allows us to use extra information on the real roots of $f(\alpha, y) \in \mathbb{R}[y]$ and $g(\alpha, y) \in \mathbb{R}[y]$ for candidate validation. We replace the tests based on interval evaluation (see page 5) by a test based on the *bitstream Descartes* isolator [15] (Bdc for short). This method allows us to isolate the real roots of a polynomial with "bitstream" coefficients, that is, coefficients that can be approximated to arbitrary precision. Bdc starts from an interval guaranteed to contain all real roots of a polynomial, and proceeds with interval subdivisions giving rise to a *subdivision tree*. Accordingly, the approximation precision for coefficients is increased in each step of the algorithm. Each leaf of the tree is associated with an interval and stores an upper and a lower bound on the number of real roots within this interval based on Descartes' Rule of Signs. An interval is not further subdivided when both bounds equal 0, where the interval is discarded, or 1, where we have found an *isolating interval*. Isolating intervals can be refined to arbitrary precision. We remark that Bdc terminates if all real roots are simple. Otherwise, intervals which contain a multiple root are further refined but never certified to contain a root.

In our algorithm, we apply Bdc to the polynomials $f(\alpha, y)$ and $g(\alpha, y)$. Eventually, intervals that do not share a common root of both polynomials will be discarded. This property is essential for our "filtered" algorithm: a candidate box $B(\alpha, \beta)$ can be rejected as soon as the associated y-interval $I(\beta)$ does not overlap with *at least* one of the isolating intervals associated with $f(\alpha, y)$ or $g(\alpha, y)$; see Figure 1 (a).

Grouping candidates along a fiber $x = \alpha$ also enables us to use *combinatorial* tests to discard or to certify them. First, when the number of certified solutions reaches mult(α), the remaining candidates are automatically discarded because each real solution contributes at least once to α's multiplicity as a root of $R^{(y)}$ (cf. Theorem 3.1). Second, if α is not a root of the greatest common divisor $h^{(y)}(x)$ of the leading coefficients of f and g and mult(α) is odd and all except one candidate along the fiber are discarded, then the remaining candidate must be a real solution. This is because complex roots come in conjugate pairs and, thus, do not change the parity of mult(α). We remark that, in case where the system (2.1) is in *generic position* and the multiplicities of all roots of R are odd, the combinatorial test already suffices to certify all solutions without the need to apply our inclusion predicate from Section 3.

Now, suppose that, after the combinatorial test, there are several candidates left along a fiber. For instance, the latter can indicate the presence of *covertical* solutions. In this case, before using the inclusion predicate, we can apply the aforementioned filters in *horizontal* direction. More precisely, we construct the lists of unvalidated candidates sharing the same y-coordinate β and process them along a horizontal fiber. For this step, we initialize the bitstream trees for $f(x, \beta)$ and $g(x, \beta)$ and proceed in exactly the same way as done for vertical fibers; see Figure 1 (b). Candidates that still remain undecided after all tests are processed by considering our inclusion predicate. In Section 5, where we next examine the efficiency of our filters, we will refer to this procedure as the *bidirectional* filter.

5 Implementation & Experiments

We have implemented our algorithm as a prototypical package of Cgal.[10] As throughout the library we follow a *generic programming paradigm* that, for instance, enables us to easily exchange the number types used or the method to isolate the roots of a polynomial without altering the main structure of the implementation.

In our experiments, we have used the number types provided by Gmp 4.3.1 and fast polynomial GCD from Ntl 5.5 library.[13] All experiments have been run on 2.8GHz 8-Core Intel Xeon W3530 with 8 MB of L2 cache

[9]Here we are not concerned with the fact that Vandemonde systems are notoriously ill-conditioned since all operations are performed in a finite field.

[10]Computational Geometry Algorithms Library, `www.cgal.org`.

[13]Gmp: `http://gmplib.org`, Ntl: `http://www.shoup.net/ntl`

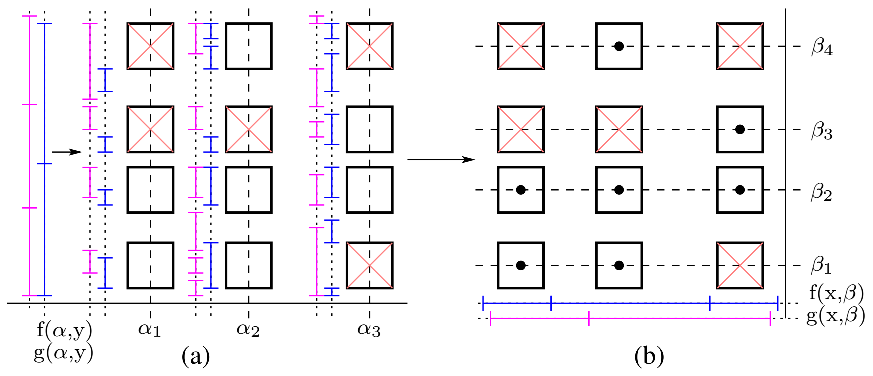

Figure 1: **(a)** Intervals containing the roots of $f(\alpha, y)$ and $g(\alpha, y)$ are refined until they either do not overlap or are fully included in candidate boxes. In the former case, the boxes can be discarded. **(b)** Unvalidated candidates are passed to *bidirectional* filter which runs bitstream isolation in another direcion

under Linux platform. For the GPU-part of the algorithm, we have used the GeForce GTX480 graphics processor (Fermi Core). We compared our approach to the bivariate version of ISOLATE (based on RS by Fabrice Rouillier[14]) and LGP by Xiao-Shan Gao et al.[15] Both were interfaced using Maple 13. We remark that, for the important substep of isolating the real roots of the elimination polynomial, all three contestants (including our implementation) use the highly efficient implementation provided by RS.

Our tests consist of two parts: In the first part, we consider "special" curves (and their derivative w.r.t. y-variable) selected in the aim of challenging different parts of the algorithm and showing the efficiency of the filtering techniques given in Section 4.2. These curves, for instance, have many singularities or high-curvature points which requires many candidates to be tested along each vertical line, or prohibit the use of special filters. Descriptions of the considered curves and corresponding timings are listed in Table 1 and Table 2, respectively. In the second part of our experiments, we study the performance of the BISOLVE on random polynomials with increasing total degrees and coefficient bit-lengths. We refer the reader to Table 3 for the corresponding timings.

In columns 4–8 of Table 2, the experiments for our algorithm are given with all filters set on (BS+ALLFILTERS), with bitstream and combinatorial filter (BS+BSTR+COMB), with bitstream filter only (BS+BSTR) and with all filters set off (BS). For BISOLVE, we report timings respectively with and without GPU resultant algorithm. For the remaining configurations we show only the timings using GPU resultants. CPU-based timings can easly be obtained by taking the difference between BISOLVE-columns.

One can observe that our algorithm is generally superior to ISOLATE and LGP even if the filters are not used. By comparing columns 5–8 of Table 2, one can see that filtering sometimes results in a significant performance improvement. The *combinatorial* test is particularly useful when the defining polynomials of the system (2.1) have large degrees and/or large coefficient bit-length while at the same time the number of covertical or singular solutions is small compared to the total number of candidates being checked. The *bidirectional* filter is advantageous when the system has covertical solutions in one direction (say along y-axis) which are *not* cohorizontal. This is essentially the case for challenge_12, cov_sol_20 and spider.

Another strength of our approach relates to the fact that the amount of symbolic operations is crucially reduced. Hence, when the time for computing resultants is dominating, the GPU-based algorithm offers a speed-up by the factor of 2-5 over the version with default resultant implementation. It is also worth mentioning that both ISOLATE and LGP benefit from the *fast resultant computation* available in Maple while CGAL's default resultant computation[16] is generally *much slower* than that of Maple. As a result, there is a large discrepancy in columns 4 and 5 for BISOLVE.

Table 3 lists timings for experiments with random curves. Each instance consists of five curves of the same degree (9 or 15, dense or sparse) and we report the average time to compute the solutions for one of all ten pairs of curves. In order to analyze the influence of the coefficients' bit-lengths, we multiplied each curve by

[14]RS: `http://www.loria.fr/equipes/vegas/rs`
[15]LGP: `http://www.mmrc.iss.ac.cn/~xgao/software.html`
[16]Authors are indebted to CGAL developers working on resultants.

Instance	Description
L4_circles	circles w.r.t. L4-norm, clustered solutions
curve_issac	a curve appeared in [8]
tryme	covertical solutions, many candidates to check
large_curves	large number of solutions
degree_6_surf	silhouette of an algebraic surface, covertical solutions in both directions
challenge_12*	many candidate solutions to be checked
SA_4_4_eps*	singular points with high tangencies, displaced
FTT_5_4_4*	many non-rational singularities
dfold_10_6*	a curve with many half-branches
cov_sol_20	covertical solutions
mignotte_xy	a product of x/y- Mignotte polynomials, displaced; many clustered solutions
spider	degenerate curve, many clustered solutions
hard_one	vertical lines as component of one curve, many candidates to test
grid_deg_10	large coefficients, curve in generic position
huge_cusp	large coefficients, high-curvature points
cusps_and_flexes	high-curvature points
L6_circles	4 circles w.r.t. L6-norm, clustered solutions
ten_circles	set of 10 random circles multiplied together, rational solutions
curve24	curvature of degree 8 curve, many singularities
compact_surf	silhouette of an algebraic surface, many singularities, isolated solutions
13_sings_9	large coefficients, high-curvature points
swinnerston_dyer	covertical solutions in both directions
challenge_12_1*	many candidate solutions to be checked
SA_2_4_eps*	singular points with high tangencies, displaced
spiral29_24	taylor expansion of a spiral intersecting a curve with many branches, many candidates to check

Table 1: Description of the curves used in the first part of experiments. In case only a single curve given, the second curve is taken to be the first derivative w.r.t. y-variable. Curves marked with a star (*) are given in [26].

2^k with $k \in \{128, 512, 2048\}$ and increased the constant coefficient by one. Since the latter operation constitutes only a small perturbation of the vanishing set of the input system, the number of solutions remains constant while the content of the polynomials' coefficients also stays trivial. We see that the bidirectional filtering is not of any advantage because the system defined by random polynomials is unlikely to have covertical solutions. However, in this case, most candidates are rejected by the combinatorial check, thereby omitting (a more expensive) test based on Theorem 3.4. This results in a clear speed-up over a "non-filtered" version. Also, observe that GPU-BISOLVE is not vulnerable to increasing the bit-length of coefficients while this becomes critical for ISOLATE's and LGP's performance. We have also observed that, for our filtered versions, the time for the validation step is almost independent of the bit-lengths.

We omit experiments to refine the solution boxes to certain precision as this matches the efficiency of QIR due to the fact that we have algebraic descriptions for the solutions' x- and y-coordinates.

All defining polynomials are archived online at `http://www.mpi-inf.mpg.de/departments/d1/projects/Geometry/BisolveDatasetAlenex2011.zip`.

6 Summary and Outlook

We propose an exact and complete method to isolate the real solutions of a bivariate polynomial system. Our algorithm is designed to reduce the number of purely symbolic operations as much as possible. Eventually, only resultant and gcd computation are still needed. By transferring the resultant computation to the GPU, we are able to remove a major bottleneck of elimination approaches. In order to further improve our implementation, we aim to outsource the square-free factorization to the GPU as well, a step which seems to be feasible since factorization is also well suited for a "divide-conquer-combine" modular approach. Since our initial motivation was to speed up the topology and arrangement computation for algebraic curves and surfaces, we plan to extend our method towards this direction. Another promising algorithm has been recently proposed by Fabrice Rouillier in [30]. It is based on computing rational univariate representations by means of computing a subresultant sequence. We expect its implementation to be available soon and aim for a comparison with our approach. Furthermore, it would be interesting to extend our algorithm to handle higher dimensional sys-

			BS+ALLFILTERS		BS+BSTR+COMB	BS+BSTR	BS	ISOLATE	LGP
Instance	y-degree	#sols	CPU	GPU	GPU			Maple	Maple
L4_circles	16	17	2.74	**1.68**	1.52	1.71	0.68	1.20	7.40
curve_issac	15	18	4.30	**3.21**	2.70	3.47	1.84	70.91	3.54
tryme	24, 34	20	98.56	**29.31**	31.63	89.83	89.86	167.81	176.86
large_curves	24, 19	137	110.20	**91.15**	90.82	376.71	377.55	501.52	138.35
degree_6_surf	42	13	149.33	**17.46**	16.50	18.63	62.18	timeout	133.77
challenge_12	40	99	108.74	**23.07**	27.66	27.20	195.76	41.13	40.86
SA_4_4_eps	33	2	155.92	**2.83**	2.85	3.81	8.57	296.02	56.30
FTT_5_4_4	40	62	73.89	**17.58**	20.92	20.99	111.22	timeout	199.92
dfold_10_6	32	21	26.20	**4.80**	3.12	3.19	3.54	3.14	3.84
cov_sol_20	20	8	27.25	**12.36**	36.10	42.81	52.16	762.80	175.85
mignotte_xy	32	30	545.88	**438.38**	440.64	986.68	1021.50	timeout	timeout
spider	28	38	389.06	**81.63**	87.44	135.15	314.56	timeout	timeout
hard_one	27, 6	46	8.17	**6.95**	6.96	12.44	12.09	25.20	20.00
grid_deg_10	10	20	4.05	**1.63**	1.64	3.22	3.01	106.95	3.16
huge_cusp	8	24	33.24	**21.43**	21.15	26.97	26.47	768.56	119.03
cusps_and_flexes	9	20	2.31	**1.37**	1.28	1.70	1.38	28.42	2.73
L6_circles	24	18	25.00	**6.21**	5.68	6.88	5.08	46.61	52.79
ten_circles	20	45	10.51	**7.64**	4.63	4.93	2.57	5.22	5.24
curve24	24	28	41.26	**16.61**	16.66	118.91	115.62	49.69	41.96
compact_surf	18	57	19.01	**6.53**	5.98	5.85	23.56	timeout	12.31
13_sings_9	9	35	3.38	**2.39**	2.23	2.98	2.41	28.60	2.97
swinnerston_dyer	40	63	50.32	**22.60**	22.53	22.33	56.46	71.00	28.47
challenge_12_1	30	99	24.67	**9.17**	9.89	9.44	41.84	41.13	40.86
SA_2_4_eps	17	6	6.71	**0.56**	0.59	0.70	1.66	7.83	4.90
spiral29_24	29, 24	51	80.37	**35.34**	35.13	290.35	286.79	144.79	84.97

Table 2: Experiments for the curves listed in Table 1. Execution times are in seconds, including resultant computations. BISOLVE-GPU: our approach with GPU-resultants; BISOLVE-CPU: our approach with CGAL's CPU-resultants; ISOLATE and LGP use Maple's implementation for the resultant computation. Bold face indicates default setup for BISOLVE and "timeout" a running time > 1500 sec.

tems or complex solutions. Finally, we would like to investigate in hybrid methods such as the combination of a numerical complex root solver and an exact post certification method serving as an additional filter in the validation step (in the spirit of [38, 33]). We are convinced that most of the candidate boxes could be treated even more efficiently by the use of such methods. We claim that, eventually, the total costs for solving a bivariate system should *only* be dominated by those of the root isolation step for the elimination polynomial. For many instances, our experiments already hint to the latter claim. We aim to further improve our implementation to show this behavior for all instances and to provide a proof in terms of complexity as well.

Acknowledgments

We thank Michael Kerber and the anonymous reviewers for their constructive and detailed criticism that have helped to improve the quality and exposition of this contribution.

References

[1] J. Abbott. Quadratic interval refinement for real roots. Poster presented at the 2006 Int. Symp. on Symb. and Alg. Comp. (ISSAC 2006).

[2] L. Alberti, B. Mourrain, and J. Wintz. Topology and Arrangement Computation of Semi-Algebraic Planar Curves. *Computer Aided Geometric Design*, 25(8):631–651, 2008.

[3] M. E. Alonso, E. Becker, M.-F. Roy, and T. Wörmann. *Zeros, multiplicities, and idempotents for zero-dimensional systems*, pages 1–15. Birkhauser Verlag, Basel, Switzerland, Switzerland, 1996.

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

[5] E. Berberich, M. Kerber, and M. Sagraloff. An efficient algorithm for the stratification and triangulation of algebraic surfaces. *Computational Geometry: Theory and Applications*, 43:257–278, 2010. Special issue on SoCG'08.

[6] M. Burr, S. W. Choi, B. Galehouse, and C. K. Yap. Complete subdivision algorithms, II: Isotopic Meshing of Singular Algebraic Curves. In *ISSAC'08:Proc.*

Density of			avg.	BS+allfilters		BS+bstr+comb	BS+bstr	BS	Isolate	Lgp
polynomials	y-degree	shift	#sols	CPU	GPU	GPU			Maple	Maple
dense	9,9	-	5.6	0.30	**0.21**	0.22	0.21	0.19	0.33	0.24
		128		0.48	**0.15**	0.16	0.15	0.49	0.66	0.93
		512		2.22	**0.31**	0.31	0.31	2.03	1.51	3.33
		2048		16.47	**2.07**	2.06	2.07	13.86	7.48	102.37
dense	15,15	-	5.0	1.82	**0.71**	0.70	0.71	1.64	6.85	3.88
		128		6.02	**0.69**	0.67	0.67	3.74	14.66	8.31
		512		32.18	**1.48**	1.45	1.48	14.35	38.27	22.36
		2048		251.07	**8.97**	8.94	8.97	94.09	141.69	102.36
sparse	9,9	-	4.5	0.10	**0.07**	0.07	0.07	0.09	0.07	0.22
		128		0.14	**0.08**	0.08	0.07	0.21	0.20	0.57
		512		0.46	**0.16**	0.15	0.15	0.85	0.70	1.74
		2048		3.11	**0.84**	0.84	0.84	5.86	5.40	7.38
sparse	15,15	-	3.8	0.65	**0.36**	0.36	0.36	0.66	0.99	1.24
		128		1.55	**0.46**	0.47	0.46	1.50	4.68	3.51
		512		7.70	**1.55**	1.55	1.55	6.80	16.01	11.93
		2048		58.97	**13.45**	13.38	13.46	51.14	132.76	74.03

Table 3: Averaged running times for 10 pairs of curves defined by random polynomials of degree 9 and 15 with increasing bit-lengths (given by **shift** parameter). For description of configurations, see Table 2.

of the 2008 Int. Symp. on Symbolic and Algebraic Computation, pages 87–94. ACM press, 2008.

[7] J. Cheng, S. Lazard, L. Penaranda, M. Pouget, F. Rouillier, and E. Tsigaridas. On the topology of planar algebraic curves. In *SCG '09: Proc. of the 25th Annual Symposium on Computational Geometry*, pages 361–370, New York, NY, USA, 2009. ACM.

[8] J.-S. Cheng, X.-S. Gao, and J. Li. Root isolation for bivariate polynomial systems with local generic position method. In *ISSAC '09*, pages 103–110, New York, NY, USA, 2009. ACM.

[9] G. E. Collins. The calculation of multivariate polynomial resultants. In *SYMSAC '71*, pages 212–222. ACM, 1971.

[10] G. E. Collins and A. G. Akritas. Polynomial real root isolation using descarte's rule of signs. In *SYMSAC '76: Proceedings of the third ACM symposium on Symbolic and algebraic computation*, pages 272–275, New York, NY, USA, 1976. ACM.

[11] D. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry*. Springer, New-York, 1998.

[12] D. I. Diochnos, I. Z. Emiris, and E. P. Tsigaridas. On the asymptotic and practical complexity of solving bivariate systems over the reals. *J. Symb. Comput.*, 44(7):818–835, 2009.

[13] A. Eigenwillig and M. Kerber. Exact and Efficient 2D-Arrangements of Arbitrary Algebraic Curves. In S.-H. Teng, editor, *SODA'08: Proceedings of the 19th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 122–131. SIAM, 2008.

[14] A. Eigenwillig, M. Kerber, and N. Wolpert. Fast and Exact Geometric Analysis of Real Algebraic Plane Curves. In C. W. Brown, editor, *Proceedings of the 2007 International Symposium on Symbolic and Algebraic Computation (ISSAC 2007)*, pages 151–158. ACM press, 2007.

[15] A. Eigenwillig, L. Kettner, W. Krandick, K. Mehlhorn, S. Schmitt, and N. Wolpert. A Descartes algorithm for polynomials with bit-stream coefficients. In *8th International Workshop on Computer Algebra in Scientific Computing (CASC 2005)*, volume 3718 of *LNCS*, pages 138–149, 2005.

[16] P. Emeliyanenko. A complete modular resultant algorithm targeted for realization on graphics hardware. In *PASCO '10*, pages 35–43, New York, NY, USA, 2010. ACM.

[17] P. Emeliyanenko. Modular Resultant Algorithm for Graphics Processors. In *ICA3PP '10*, pages 427–440, Berlin, Heidelberg, 2010. Springer-Verlag.

[18] J. V. Z. Gathen and J. Gerhard. *Modern Computer Algebra*. Cambridge University Press, New York, NY, USA, 2003.

[19] K. Geddes, S. Czapor, and G. Labahn. *Algorithms for computer algebra*. Kluwer Academic Publishers, Boston/Dordrecht/London, 1992.

[20] L. Gonzalez-Vega and I. Necula. Efficient Topology Determination of Implicitly Defined Algebraic Plane Curves. *Comput. Aided Geom. Des.*, 19(9):719–743, 2002.

[21] H. Hong. An efficient method for analyzing the topology of plane real algebraic curves. *Mathematics and Computers in Simulation*, 42(4-6):571 – 582, 1996. Sybolic Computation, New Trends and Developments.

[22] I. C. F. Ipsen and R. Rehman. Perturbation bounds for determinants and characteristic polynomials. *SIAM J. Matrix Anal. Appl.*, 30(2):762–776, 2008.

[23] T. Kailath and S. Ali. Displacement structure: theory and applications. *SIAM Review*, 37:297–386, 1995.

[24] M. Kerber. On the complexity of reliable root approximation. In V. P. Gerdt, E. W. Mayr, and E. V. Vorozhtsov, editors, *CASC*, volume 5743 of *Lecture Notes in Computer Science*, pages 155–167. Springer,

2009.

[25] H. Kobayashi, T. Fujise, and A. Furukawas. Solving systems of algebraic equations by a general elimination method. *J. Symb. Comput.*, 5(3):303–320, 1988.

[26] O. Labs. A list of challenges for real algebraic plane curve visualization software. In I. Z. Emiris, F. Sottile, and T. Theobald, editors, *Nonlinear Computational Geometry*, volume 151 of *The IMA Volumes in Mathematics and its Applications*, pages 137–164. Springer New York, 2010.

[27] B. Mourrain and J. P. Pavone. Subdivision methods for solving polynomial equations. *J. Symb. Comput.*, 44(3):292–306, 2009.

[28] S. Petitjean. Algebraic Geometry and Computer Vision: Polynomial Systems, Real and Complex Roots. *J. Math. Imaging Vis.*, 10(3):191–220, 1999.

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

[30] F. Rouillier. On solving systems of bivariate polynomials. In K. Fukuda, J. van der Hoeven, M. Joswig, and N. Takayama, editors, *ICMS*, volume 6327 of *Lecture Notes in Computer Science*, pages 100–104. Springer, 2010.

[31] F. Rouillier and P. Zimmermann. Efficient isolation of polynomial's real roots. *J. Comput. Appl. Math.*, 162(1):33–50, 2004.

[32] S. Rump. Verified bounds for singular values, in particular for the spectral norm of a matrix and its inverse. 2010. submitted, http://www.ti3.tu-harburg.de/paper/rump/Ru10a.pdf.

[33] S. M. Rump. Ten methods to bound multiple roots of polynomials. *J. Comput. Appl. Math.*, 156(2):403–432, 2003.

[34] M. Sagraloff, M. Kerber, and M. Hemmer. Certified complex root isolation via adaptive root separation bounds. In M. Suzuki, H. Hong, H. Anai, C. Yap, Y. Sato, and H. Yoshida, editors, *The Joint Conference of ASCM 2009 and MACIS 2009*, volume 22 of *MI Lecture Note Series*, pages 151–166, Fukuoka, Japan, December 2009. Math-for-Industry (MI), COE.

[35] M. Sagraloff and C. K. Yap. An efficient and exact subdivision algorithm for isolating complex roots of a polynomial and its complexity analysis. Draft, unpublished, available at `http://www.cs.nyu.edu/exact/doc/complex.pdf`, 2009.

[36] R. Seidel and N. Wolpert. On the exact computation of the topology of real algebraic curves. In *Proceedings of the 21st Annual ACM Symposium on Computational Geometry (SCG 2005)*, pages 107–115, 2005.

[37] A. J. Sommese and C. W. Wampler. *The Numerical Solution of Systems of Polynomials Arising in Engeneering and Science.* World Scientific, Singapore, 2005.

[38] A. W. Strzebonski. Cylindrical algebraic decomposition using validated numerics. *Journal of Symbolic Computation*, 41:1021–1038, 2006.

[39] B. Sturmfels. *Solving systems of polynomial equations*, volume 97 of *Regional conference series in mathematics.* AMS, Providence, RI, 2002.

[40] The CGAL Project. *CGAL User and Reference Manual.* CGAL Editorial Board, 3.7 edition, 2010. http //www.cgal.org/.

[41] C. K. Yap. *Fundamental Problems in Algorithmic Algebra.* Oxford University Press, 2000.