On the topology of real algebraic plane curves

Jinsan Cheng* Sylvain Lazard* Luis Peñaranda*
Marc Pouget* Fabrice Rouillier† Elias Tsigaridas‡

May 5, 2010

Abstract

We revisit the problem of computing the topology and geometry of a real algebraic plane curve. The topology is of prime interest but geometric information, such as the position of singular and critical points, is also relevant. A challenge is to compute efficiently this information for the given coordinate system even if the curve is not in generic position.

Previous methods based on the cylindrical algebraic decomposition use sub-resultant sequences and computations with polynomials with algebraic coefficients. A novelty of our approach is to replace these tools by Gröbner basis computations and isolation with rational univariate representations. This has the advantage of avoiding computations with polynomials with algebraic coefficients, even in non-generic positions. Our algorithm isolates critical points in boxes and computes a decomposition of the plane by rectangular boxes. This decomposition also induces a new approach for computing an arrangement of polylines isotopic to the input curve. We also present an analysis of the complexity of our algorithm. An implementation of our algorithm demonstrates its efficiency, in particular on high-degree non-generic curves.

1 Introduction

Let $\mathcal{C}$ be a real algebraic plane curve defined in a Cartesian coordinate system by a bivariate polynomial f with rational coefficients, *i.e.*, $\mathcal{C} = \{(x, y) \in \mathbb{R}^2 \mid f(x, y) = 0\}$ with $f \in \mathbb{Q}[x, y]$. We consider the problem of computing the topology of $\mathcal{C}$ with additional geometric information associated to the given coordinate system. By computing the topology of $\mathcal{C}$, we mean to compute an arrangement of polylines $\mathcal{G}$, that is topologically equivalent to $\mathcal{C}$ (see Figure 1). Note that this arrangement of polylines $\mathcal{G}$ is often identified to a graph embedded in the plane, where the vertices can be place at infinity and the edges are straight line segments. We first define formally what we mean by topologically equivalent, before discussing the additional geometric information we consider and their relevance.

Two curves $\mathcal{C}$ and $\mathcal{G}$ of the Euclidean plane are said to be (ambient) isotopic if there exists a continuous map $F : \mathbb{R}^2 \times [0, 1] \longrightarrow \mathbb{R}^2$, such that $F_t = F(\cdot, t)$ is a homeomorphism for any $t \in [0, 1]$, F_0 is the identity of $\mathbb{R}^2$ and $F_1(\mathcal{C}) = \mathcal{G}$. This notion formalizes the idea that one can deform one

*LORIA (INRIA, CNRS, Nancy Université) and INRIA Nancy - Grand Est, Nancy, France. Firstname.Name@loria.fr

†LIP6 (Université Paris 6, CNRS) and INRIA Paris-Rocquencourt, Paris, France. Firstname.Name@lip6.fr

‡Dept. of Computer Sciece, Århus University, Denmark. Firstname.Lastname@gmail.com

(a) (b) (c)

Figure 1: (a) Curve of equation $16\,x^5 - 20\,x^3 + 5\,x - 4\,y^3 + 3\,y = 0$ plotted in MAPLE and (b) its isotopic graph computed by ISOTOP.

curve to the other by a deformation of the whole plane. Isotopy is stronger than homeomorphy, for instance, (i) two nested loops and (ii) two non-nested loops are not isotopic.[1]

We now discuss the relevance of adding geometric information to the graph $\mathcal{G}$. From the topological point of view, the graph $\mathcal{G}$ must contain vertices that correspond to self-intersections and isolated points of the curve. However, in order to avoid separating such relevant points from other singularities (*e.g.*, cusps), all *singular points* of $\mathcal{C}$, that is, points at which the tangent is not well defined, are chosen to be vertices of the graph.

While singular points are needed for computing the topology of a curve, the extreme points of a curve are also very important for representing its geometry. Precisely, the *extreme points* of $\mathcal{C}$ for a particular direction, say the direction of the x-axis, are the non-singular points of $\mathcal{C}$ at which the tangent line is vertical (*i.e.*, parallel to the y-axis); the extreme points in the direction of the x-axis are called *x-extreme*. These extreme points are crucial for various applications and, in particular, for computing arrangements of curves by a standard sweep-line approach [EK08]. Of course, one can theoretically compute an arrangement of algebraic curves by computing the topology of their product. However, this approach is obviously highly inefficient compared to computing the topology of each input curve and, only then, computing the arrangement of all the curves with a sweep-line algorithm. Note that the x-extreme and singular points of $\mathcal{C}$ form together the *x-critical points* of the curve (the x-coordinates of these points are exactly the positions of a vertical sweep line at which there may be a change in the number of intersection points with $\mathcal{C}$).

It is thus useful to require that all the x-critical points of $\mathcal{C}$ are vertices of the graph we want to compute. To our knowledge, almost all methods for computing the topology of a curve compute the *critical points* of the curve and associate corresponding vertices in the graph. (Refer to [AMW08, BSGY08] for recent subdivision methods that avoid the computation of non-singular critical points.) However, it should be stressed that almost all methods do not necessarily compute the critical points for the specified x-direction. Indeed, when the curve is not in *generic position*, that is, if

[1] Note that in two dimensions, the notion of ambient isotopy is equivalent to the notion of ambient homeomorphy, that is, to the existence of an orientation preserving homeomorphism of $\mathbb{R}^2$ that maps $\mathcal{C}$ onto $\mathcal{G}$ [Bir75, Thm. 4.4 p.161].

two x-critical points have the same x-coordinate or if the curve admits a vertical asymptote, most algorithms shear the curve so that the resulting curve is in generic position. This is, however, an issue for several reasons. First, determining whether a curve is in generic position is not a trivial task and it is time consuming [GVEK96, SF90]. Second, if one wants to compute arrangements of algebraic curves with a sweep-line approach, the extreme points of all the curves have to be computed for the same direction. Finally, if the coordinate system is sheared, the polynomial of the initial curve is transformed into a dense polynomial, which slows down, in practice, the computation of the critical points.

In this paper, given a curve $\mathcal{C}$ which is not necessarily in generic position, we aim at computing efficiently its topology, including all the critical points for the specified x-direction. In other words, we want to compute an arrangement of polylines that is isotopic to $\mathcal{C}$ and whose vertices include the x-critical points of $\mathcal{C}$. In terms of efficiency, our primary goal is the practical efficiency rather than worst-case complexity. In particular, we want to avoid computations with non-rational algebraic numbers or, equivalently in this context, algebraic computations such as Sturm sequences, Sturm-Habicht sequences (which are a generalization of Sturm sequences, with better specialization properties [GVLRR89]), and sub-resultant sequences.

We review below previous work on the problem and then present our contributions.

Previous Work. There have been many papers addressing the problem of computing the topology of algebraic plane curves (or closely related problems) defined by a bivariate polynomial with rational coefficients [AMW08, AM88, CR88, BPR06, EKW07, Fen92, GVEK96, GVN02, Hon96, MPS$^+$06, Sak91, Bro01, ACM84, MC02, Str06, DET07]. Most of the algorithms assume generic position for the input curve. As mentioned above, this is without loss of generality since we can always shear a curve into generic position [SF90, GVEK96] but this has a substantial negative impact on the time computation. All these algorithms perform the following phases. (1) Project the x-critical points of the curve on the x-axis using sub-resultants sequences, and isolate the real roots of the resulting univariate polynomial in x. This gives the x-coordinates of all the x-critical points. (2) For each such value x_i, compute the intersection points between the curve $\mathcal{C}$ and the vertical line $x = x_i$. (3) Through each of these points, determine the number of branches of $\mathcal{C}$ coming from the left and going to the right. (4) Connect all these points appropriately.

The main difficulty in all these algorithms is to compute efficiently all the critical points in Phase 2 because the x-critical values in Phase 1 are, a priori, non-rational thus computing the corresponding y-coordinates in Phase 2 amounts, in general, to solving a univariate polynomial with non-rational coefficients and at least a multiple root (corresponding to the critical point). To this end, most algorithms [AM88, CR88, BPR06, Fen92, GVEK96, GVN02, Hon96, MPS$^+$06, Sak91, ACM84, Str06] rely heavily on computations with real algebraic numbers, Sturm sequences or sub-resultant sequences.

An approach using a variant of sub-resultant sequences for computing the critical points in Phase 2 was proposed by Hong [Hon96]. He computed this way (xy-parallel) boxes with rational endpoints and separating the critical points. Counting the branches in Phase 3 can then be done by intersecting the boundary of the boxes with the curve which only involves univariate polynomials with rational coefficients. This approach, see also [ACM84, MC02, Bro01] and the software package Cad2D[2], does not assume that the curve is in generic position.

In a more recent paper, González-Vega and Necula [GVN02] use Sturm-Habicht sequences which allow them to express the y-coordinate of the each critical point as a rational function of its x-

[2]http://www.cs.usna.edu/∼qepcad/B/user/cad2d.html

coordinate. They implemented their algorithm in MAPLE with symbolic methods modulo the fact that the algebraic coefficients of the polynomials in Phase 2 are approximated in fixed-precision arithmetic. The algorithm takes as a parameter the initial precision for the floating-point arithmetic and recursively increases the precision. This approach is however not certified in the case where the curve is not in generic position because the algorithm checks for the equality of pairs of polynomials whose coefficients are evaluated (incorrect results have been reported in [SW05]). Note that there exists one variant of González-Vega and Necula algorithm that handles, without shearing, curves that are not in generic position [MPS$^+$06]. This approach, however, requires substantial additional time-consuming symbolic computations such as computing Sturm sequences.

Recently, Seidel and Wolpert [SW05] presented an alternate approach for computing the critical points avoiding most costly algebraic computations but to the expense of computing several projections of the critical points. They project, in Phase 1, the x-critical points on both x and y-axes and also on a third random axis. After isolating the roots on each axis, they can recover (xy-parallel) boxes with rational endpoints that contain each exactly one critical point. From there, all computations only involve rational numbers but they, however, still need to compute Sturm-Habicht sequences for refining the boxes containing the singular points until each box interests only the branches of the curve incident to the singular point. Their approach assumes that the curve is in generic position by a pre-processing phase in which the curve is sheared if needed. They also present a MAPLE implementation, INSULATE, which is an implementation of a certified algorithm for curves in arbitrary positions. Note that their implementation does not report x-extreme points in the original system when the curved is sheared.

Even more recently, Eigenwillig et al. [EKW07] (see also [Ker06]) presented a variant of González-Vega and Necula approach, in which the roots of the polynomials with non-rational coefficients are efficiently isolated using an implementation of a variant of an interval-based Descartes algorithm [EKK$^+$05]. This variant, as [GVN02], does not assume that the curve is in generic position but detects such configurations online. More precisely, if the bit-stream Descartes algorithm is in, a sense, unlucky then, rather than refining down to a separation bound (*e.g.* [BFM$^+$01, LPY04]), the algorithm shears the input curve and starts again. Note that this approach still computes Sturm-Habicht sequences for determining the polynomials appearing in Phase 2 and the multiplicity of its multiple root. Also, if the curve is sheared to a (x', y') coordinate system, they compute extreme points both for the x'-direction and the direction corresponding to the x-axis. This approach has been implemented in C++ and is an implementation of a certified algorithm that handles curves that are not necessarily in generic positions and that reports x-extreme points for the original coordinate system.

Note finally that another approach that avoids expensive algebraic computations is to compute the critical or singular points using subdivision methods [AMW08, BSGY08]. The major drawback of these methods is that, in order to certify the results, the subdivision has, in general, to reach a separation bound (certification can also be achieved by solving algebraic systems by other means). It follows that, if no certification is required, these methods are very fast in practice, however, they can become very slow on difficult instances, if certification is required. To our knowledge, no implementation of such certified algorithm is available.

As for the complexity of the problem, the best known bound so far is $\widetilde{\mathcal{O}}(N^{12})$ [DET07], where N is essentially the maximum of the degree of f and of the maximum number of bits needed for representing the input coefficients and the notation $\widetilde{\mathcal{O}}$ denotes that the poly-logarithmic factors are omitted. This assumes that the real algebraic numbers are represented by isolating intervals.

This complexity is based on theoretically fast computations with Sturm-Habicht sequences and algorithms for univariate real root isolation.

Our contributions. We present an algorithm for computing the topology of an algebraic plane curve which is possibly in non-generic position. The algorithm handles curves in non-generic positions in the Cartesian coordinate system in which they are defined. In particular, the algorithm never shears the coordinate system, which avoids the associated extra costs discussed above.

Another specificity of our approach is that we succeed to avoid, in all cases, the computation of sub-resultant sequences and computations with algebraic numbers. Instead, we compute, in particular, Gröbner bases [Buc85]. One can argue that these are just two different tool sets, and this is true from a theoretical point of view, but we show in the experiments the benefit of our choice when computing with non-generic curves. Furthermore, the philosophy of our approach is to avoid, as much as possible, computations that are time consuming in practice. This leads to various algorithmic choices such as avoiding the computation of y-critical points and allowing the curve to intersect the top and bottom sides of boxes isolating x-extreme points (which avoids substantial subdivisions since the tangent at an extreme point is vertical).

The novelty of our algorithm mainly relies upon the use of three new ingredients for this problem. First, we use some of the state-of-the-art techniques to isolate the roots of bivariate systems, *i.e.,* we use (i) Gröbner basis computations [Fau99], (ii) Rational Univariate Representations (RUR) [Rou99] which represent the roots of the system as rational functions of the roots of a univariate polynomial, and (iii) a subdivision technique based on Descartes' rule of signs (and filtered with interval arithmetic for efficiency) for isolating the roots of the univariate polynomials [RZ03, ESY06, EMT08]. Even though this approach is well known for system solving, it was not used before for computing the topology of algebraic curves.

Second, we compute and use the multiplicities in fibers (see Definition 1) of critical points to compute the topology at singular points and to determine the connections at extreme points. For extreme points, we get these multiplicities by the RUR and a special case of a formula of Teissier [Tei73]. For singular points, we solve the system of singular points with additional constraints. Note that the overall method to compute the topology at critical points is not new, it is described in full details in [SW05], see also for example [ACM84, Hon96, Bro01, MC02, MPS$^+$06, AMW08] for closely related approaches for curves in non-generic position. The novelty appears in the way we compute multiplicities in this context; once again we avoid computing sub-resultant sequences.

Third, we present a variant of the standard combinatorial part of the algorithm for computing the topology. We compute a decomposition of the plane (which is not a cylindrical algebraic decomposition) by rectangular boxes containing at most one critical point. Since we allow crossings on the top and bottom of extreme point boxes to avoid costly refinement of boxes, the connection is not always straightforward. To achieve the connection near such points, we use the multiplicities in fibers of extreme points. Note that one advantage of this variant is that, even when the curve is in generic position, the algorithm does not require to refine these boxes until they do not overlap in x.

With these tools, our algorithm for computing an arrangement of polylines $\mathcal{G}$ isotopic to a curve $\mathcal{C}$ can be summarized as follows (see Section 3 for details). (1) Isolate the x-critical points in two dimensional rectangles, called critical boxes, using algebraic tools (Gröbner bases, RUR and Descartes' algorithm). Compute also the multiplicity of the critical points in their fibers. Refine the critical boxes until the restriction of $\mathcal{C}$ to each critical box is guaranteed to be a set of non-crossing arcs connecting the critical point to a point on the boundary of the box. (2) Compute a rectangular

decomposition of the plane by extending the vertical sides of the critical boxes either to infinity or to the first encountered critical box (see Figure 2(a)). (Note that, for visualization purposes, a geometrically accurate picture of $\mathcal{C}$ can be easily obtained by further subdividing vertically the non-critical rectangles of the decomposition.) For every edge of this decomposition, determine its intersection with $\mathcal{C}$, that is, determine separating intervals each containing exactly one intersection point. (3) The vertices of $\mathcal{G}$ consist of the x-critical points of $\mathcal{C}$ and intersections of $\mathcal{C}$ with the edges of the rectangular decomposition. For every critical box of $\mathcal{G}$, connect (with a straight line segment) the critical vertex to the vertices on the boundary of the box. For every other rectangle of the decomposition, connect the vertices on its boundary using, if needed, the multiplicity of the extreme points in the neighboring rectangles combined with a greedy approach. The output of the algorithm is an arrangement of polylines represented by the embedded graph $\mathcal{G}$. The vertices of G hence represent points whose coordinates are, in general, non-rational. Associated to each vertex, the algorithm also computes a box containing the represented point (the critical box for a critical point or the separating interval determined in Step 2 for an intersection between the curve and a wall of the rectangular decomposition). These boxes can be refined, and any choice of point (for instance, with rational coordinates) in these boxes gives a graph isotopic to the curve.

We also analyze the worst-case bit complexity of our algorithm. To the best of our knowledge, this is, for the problem considered here, the first time that the complexity of a (certified) algorithm based on refinements and approximations is analyzed. Our technique, even though not novel, could presumably also be used to analyze the complexity of the algorithms in [GVN02, Hon96, SW05, EKW07].

Finally, we ran large scale benchmarks on over a thousand of curves during several weeks for comparing the different state-of-the-art available implementations.

The rest of the paper is organized as follows. First, we recall in Section 2 some basic material, in particular, on the multiplicity of the intersection of two curves and on Rational Univariate Representations of the roots of system of (algebraic) equations. We present our algorithm in Section 3 and its complexity in Section 4. Finally, we present in Section 5 extensive experiments comparing various implementations.

2 Preliminaries

Let $\mathcal{C}$, also denoted $\mathcal{C}_f$, be a real algebraic plane curve defined by a bivariate polynomial f in $\mathbb{Q}[x, y]$. Since the geometry of the curve is not modified by taking the square-free part of f, we can assume without loss of generality that f is square-free. Note that $\mathcal{C}$ may consist of several algebraic components, that is, f is not necessarily irreducible in $\mathbb{R}[x, y]$. The algebraic components of the curve that are vertical lines (*i.e.*, lines parallel to the y-axis) can be easily computed since their abscissa correspond to the real roots of the polynomial in x obtained as the gcd of the coefficients of f seen as a polynomial in y.

Partial derivatives are denoted with subscripts: for instance, f_x denotes the derivative of f with respect to x and f_{y^k} (sometimes also f_k) denotes the k^{th} derivative with respect to y. A point $\mathbf{p} = (\alpha, \beta) \in \mathbb{C}^2$ is called x-critical if $f(\mathbf{p}) = f_y(\mathbf{p}) = 0$, singular if $f(\mathbf{p}) = f_x(\mathbf{p}) = f_y(\mathbf{p}) = 0$, and x-extreme if $f(\mathbf{p}) = f_y(\mathbf{p}) = 0$ and $f_x(\mathbf{p}) \neq 0$ (*i.e.*, it is x-critical and non-singular). Similarly are defined y-critical and y-extreme points. As x-critical and x-extreme points are more often used in the following, we often simply refer to them as critical and extreme points.

The ideal generated by polynomials $P_1, \ldots, P_i$ is denoted $\mathbb{I}(P_1, \ldots P_i)$. In the following, we

often identify the ideal and the system of equations $\{P_1 = 0, \ldots, P_i = 0\}$ (or any equivalent system induced by a set of generators of the ideal). We consider, in particular, the ideals $I_c = \mathbb{I}(f, f_y)$ and $I_s = \mathbb{I}(f, f_x, f_y)$; their roots are, respectively, the x-critical and singular points of $\mathcal{C}$.

Multiplicities. We now recall the notion of multiplicity of the roots of an ideal, then we state two lemmas using this notion for studying the local topology at critical points. Geometrically, the notion of multiplicity of intersection of two regular curves is intuitive. If the intersection is transverse, the multiplicity is one; otherwise, it is greater than one and it measures the level of degeneracy of the tangential contact between the curves. Defining the multiplicity of the intersection of two curves at a point that is singular for one of them (or possibly both) is more involved and an abstract and general concept of multiplicity in an ideal is needed.

Definition 1 ([CLO05, §4.2]). *Let I be an ideal of $\mathbb{Q}[x, y]$ and denote $\overline{\mathbb{Q}}$ the algebraic closure of $\mathbb{Q}$. To each zero (α, β) of I corresponds a local ring $(\overline{\mathbb{Q}}[x, y]/I)_{(\alpha,\beta)}$ obtained by localizing the ring $\overline{\mathbb{Q}}[x, y]/I$ at the maximal ideal $\mathbb{I}(x-\alpha, y-\beta)$. When this local ring is finite dimensional as $\overline{\mathbb{Q}}$-vector space, we say that (α, β) is an isolated zero of I and this dimension is called the* **multiplicity of (α, β) as a zero of I**.

Let $f, g \in \mathbb{Q}[x, y]$ be such that the intersection of $\mathcal{C}_f$ and $\mathcal{C}_g$ in $\mathbb{C}^2$ contains a zero-dimensional component equal to point $\mathbf{p} = (\alpha, \beta)$. Then (α, β) is an isolated zero of $\mathbb{I}(f, g)$ and its multiplicity, denoted by $Int(f, g, \mathbf{p})$, is called the **intersection multiplicity of the two curves at this point**.

We call a **fiber** *a vertical line of equation $x = \alpha$. For a point $\mathbf{p} = (\alpha, \beta)$ on the curve $\mathcal{C}_f$, we call the multiplicity of β in the univariate polynomial $f(\alpha, y)$ the* **multiplicity of p in its fiber** *and denote it as $mult(f(\alpha, y), \beta)$.*

The next lemma, due to Teissier [Tei73], relates the multiplicity of a point in a fiber with the multiplicity in the critical ideal. We will use it to deduce the multiplicity in the fiber knowing multiplicity in the ideal. More precisely, we will use the multiplicity in fibers of extreme points during the connection step of our algorithm.

Lemma 2 ([Tei73][BR90, Lemma D.3.4 p.314]). *For an x-extreme point $\mathbf{p} = (\alpha, \beta)$ of f one has*

$$mult(f(\alpha, y), \beta) = Int(f, f_y, \mathbf{p}) + 1. \tag{1}$$

To compute the local topology of the curve at a singular point, we aim at isolating the singular point in a box so that the intersection of its border and the curve determines the topology. Indeed for a small enough box, the topology is given by the connection of the singular point with all the intersections on the border. So the box shall avoid parts of the curve not connected to the singular point. Knowing the multiplicity of the singular point in the fiber enables to isolate the singular point from other crossings of the curve in this fiber. Requiring in addition that intersections with the curve only occur on the left or right sides of the box leads to the following.[3]

Lemma 3 ([SW05]). *Let $\mathbf{p} = (\alpha, \beta)$ be a real singular point of the curve $\mathcal{C}_f$ of multiplicity k in its fiber. Let B be a box satisfying (i) B contains $\mathbf{p}$ and no other x-critical point, (ii) the function f_{y^k} does not vanish on B, and (iii) the curve $\mathcal{C}_f$ crosses the border of B only on the left or the right sides. Then the topology of the curve in B is given by connecting the singular point with all the intersections on the border.*

[3]The proof is based on a recursive application of the mean value theorem stating that the roots of the derivative of a polynomial P lie between those of P.

We finally describe the algebraic tools we use for isolating the roots of univariate and bivariate ideals.

Univariate Root Isolation We need to count and isolate the real roots of univariate polynomials, possibly in a given interval. This is, in particular, needed for computing the intersections between $\mathcal{C}$ and the sides of the boxes isolating the critical points. Only polynomials with rational coefficients will be considered. The square-free part of the considered polynomials is first computed. The real roots are then isolated using recursive subdivision and the Descartes' rule of signs (see [ESY06, EMT08, RZ03] for details and [RZ03] for the way interval arithmetic can be used to speed up computations). In our implementation, we use the RS software [RS].

Rational Univariate Representation [Rou99]. In our algorithm, we need to represent solutions of zero-dimensional ideals depending on two variables by boxes containing them. We use the so-called Rational Univariate Representation (RUR) of the roots, which can be viewed as a univariate equivalent to the studied ideal. The key feature of this RUR is the ability to isolate solutions in easily refinable boxes and to compute multiplicities.

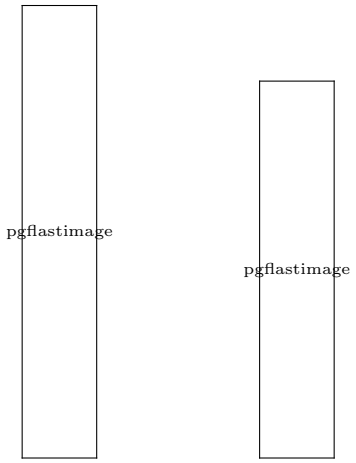
Given a zero-dimensional ideal $I = \mathbb{I}(P_1, \ldots, P_s)$ where the $P_i \in \mathbb{Q}[x_1, \ldots, x_n]$, a Rational Univariate Representation of the solutions $\mathrm{V}(I)$ is given by $F(t) = 0, x_1 = \frac{G_1(t)}{G_0(t)}, \ldots, x_n = \frac{G_n(t)}{G_0(t)}$, where $F, G_0, \ldots, G_n$ are univariate polynomials in $\mathbb{Q}[t]$ (where t is a new variable). All these univariate polynomials, and thus the RUR, are uniquely defined with respect to a given polynomial $\gamma \in \mathbb{Q}[x_1, \ldots, x_n]$ which is injective on $V(I)$; γ is called the *separating polynomial* of the RUR.[4] Note that a random degree-one polynomial in $x_1, \ldots, x_n$ is a separating polynomial with probability one. The RUR defines a bijection between the (complex and real) roots of the ideal I and those of F. Furthermore, this bijection preserves the multiplicities and the real roots. Computing a box for a solution of the system is done by isolating the corresponding root of the univariate polynomial F and evaluating the coordinate functions with interval arithmetic (see Section 4 for details on interval analysis). To refine a box, one just needs to refine the corresponding root of F and evaluate the coordinates again.

There exists several ways for computing a RUR. One can use the strategy from [Rou99] which consists of computing a Gröbner basis of I and then to perform linear algebra operations to compute a separating element as well as the full expression of the RUR. The Gröbner basis computation can also be replaced by the generalized normal form from [MT05]. There exists more or less certified alternatives such as the Geometrical resolution [GLS01] (it is probabilistic since the separating element is randomly chosen and its validity is not checked, one also loses the multiplicities of the roots) or resultant based strategies such as [KOR05]. In our implementation, we use the strategy from [Rou99] and compute Gröbner bases using the algorithm F_4 [Fau99].

3 Algorithm

Our algorithm neither our implementation require assumptions about the existence of vertical lines, but the processing of vertical lines is rather technical and, for clarity, we first describe our algorithm for curves without vertical lines and postpone the treatment of vertical lines to Section 3.2. A proof

[4] The polynomial F is the characteristic polynomial the multiplication operator by the polynomial γ, in $\mathbb{Q}[x_1, \ldots, x_n]/I$.



(a) (b)

Figure 2: (a) Example of rectangle decomposition of the plane induced by the isolating boxes (critical and asymptotes). There are boxes for the asymptote, the singular point, and the three extreme points, two of them with even multiplicities and one with odd multiplicity. (b) Possible connections involving extreme points depending on their multiplicities.

of correctness of the algorithm is presented in Section 3.3 and the algorithm is illustrated on an example in Section 3.4.

3.1 Curve without vertical lines

As discussed in Section 2, we consider, as input, a curve $\mathcal{C}$ without vertical lines and defined by a square-free polynomial $f \in \mathbb{Q}[x, y]$. In a few words, the algorithm first focuses on critical points, their rational univariate representations enable to compute multiplicities and boxes isolating each point with known topology inside the box. Then a sweep method computes a rectangular decomposition of the plane induced by the boxes of critical points. Eventually the connection is processed in all rectangles with a greedy method using multiplicities in fibers for extreme points. We describe more precisely our algorithm in six steps.

Step 1. Isolating boxes of the singular points and of the x-extreme points. As a general practical rule, the smaller the number of solutions of a system, the easier it is to work with. Hence we split the system of critical points into the system of singular points and the system of extreme points. The system of singular points is the one of critical with in addition the equation $f_x = 0$. The system of extreme points, denoted I_e, is computed by saturation. Indeed, the extreme points are critical for which $f_x \neq 0$, thus we add to the critical system the equation $1 - uf_x = 0$ with a new variable u that we eliminate afterwards. We then compute the RURs of these systems I_s and I_e and isolating boxes for the solutions (see Section 4 for details on interval analysis). We may need to refine the boxes of extreme and singular points to avoid overlaps.

Step 2. Multiplicities of critical points in fibers. For extreme points, we use the Teissier formula: the multiplicity of an extreme point in I_c is the same than in I_e because precisely f_x does not vanish at these points. The multiplicity in I_e is given by the RUR, and hence the multiplicity

of an extreme point in its fiber is this number plus one according to the equation of Lemma 2.

For singular points, we use the definition of univariate multiplicity, namely the smallest integer k such that the k^{th} derivative no longer vanishes. Let $I_{s,k}$ be the system of singular points with in addition the equations $f_{y^i} = 0$ for i from 2 to k. Hence we solve, for k increasing from 2, the systems $I_{s,k}$ until it has no solutions. At each step, a singular point which was a solution of $I_{s,k-1}$ but is no longer solution of $I_{s,k}$ has its multiplicity in fiber equal to k[5]. Note that the data of the systems $I_{s,k}$ will not be used later, they are only useful for the multiplicity computation. Theoretically, the complexity of solving these systems is analyzed in Section 4.2. In practice, as k increases, the systems have less and less solutions and hence tend to be easier to solve. Note also that the number of systems to solve is the highest multiplicity of the singular points of the curve.

Step 3. Refinement of the isolating boxes of the x-extreme points. Consider each such box, B, in turn. For each vertical or horizontal side of B, isolate its intersections with $\mathcal{C}$ and refine the box until there are two intersection points. We further refine until there is at most one crossing on the top (resp. bottom) side of B. Note that, unlike comparable algorithms, we do not require that $\mathcal{C}$ intersects the boundary of B on its vertical sides. This is important in practice because, since the curve has a vertical tangent at an x-extreme point, refining until the curve intersects the vertical side is time consuming.

Step 4. Refinement of the isolating boxes of the singular points. We refine these boxes exactly as in [SW05] (see Lemma 3) except for the way the multiplicity k of each singular point in its fiber is computed. In [SW05], k is computed using Sturm-Habicht sequences under the assumption of generic position while we deduce k as explained in Step 2. Then, as in [SW05], every box is refined until the evaluation of f_{y^k} with interval arithmetic does not contain 0 (see Section 4 for details on interval analysis). Further refine the x-coordinates of the box until $\mathcal{C}$ only intersects the vertical boundary of the box.

Step 5. Vertical asymptotes. To determine the topology of a curve, it is required to know how many branches are going to infinity. However, it is not required, in general, to know which branch is related to which asymptote. Nevertheless, for our next connection step, we need to determine for each vertical asymptote which branches are related to it and if they are on its left or its right.

The x-coordinates of vertical asymptotes are the roots of the leading coefficient $V_a(x)$ of the polynomial $f(x, y)$ considered as a polynomial in y. To deal with an asymptote $x = \alpha$, the idea is, informally, to isolate the point (α, ∞) in a box $[a, b] \times [M, ...\infty..., -M]$ whose vertical sides do not intersect the curve $\mathcal{C}$. Moreover, we want that every branch that intersects a horizontal side of the box is a branch going to $\pm\infty$ with this asymptote (see Figure 2(a)). First, compute an upper bound M_y on the absolute value of the y-coordinates of the y-critical points (this is of course done without computing these critical points, but only the discriminant with respect to y and an upper bound of the roots of this univariate polynomial). Compute also a bound M_x on the absolute value of the y-coordinates of the x-critical points (for which we have already computed boxes). Isolate the roots of the polynomial V_a, hence each root α has an isolating interval $[a, b]$. Substitute $x = a$ (resp. $x = b$) in f and deduce an upper bound, M_1, on the absolute value of the y-coordinates of the intersection of $\mathcal{C}$ and $x = a$ (resp. $x = b$). Set $M = \max(M_1, M_x, M_y)$. Then, a branch crossing the segment $]a, b[\times M$ (resp. $]a, b[\times -M$) goes to $+\infty$ (resp. $-\infty$) with asymptote $x = \alpha$. Finally,

[5] We also refer the interested reader to a more elegant way to compute the multiplicity in fibers with the Teissier formula [CLP+08]. Experimentally, it appears that this alternative was less efficient because even if it usually needs to work with less systems, these systems are larger (i.e. with more solutions).

we determine whether a given branch is to the left or to the right of the asymptote by comparing the x-coordinates of the asymptote and the crossing point.

Step 6. Connections. For simplicity, all the boxes computed above are called critical boxes and the points at infinity on vertical asymptotes are also called critical. First compute, with a sweep-line algorithm, the vertical rectangular decomposition obtained by extending the vertical sides of the critical boxes either to infinity or to the first encountered critical box (see Figure 2(a)). On each of the edges of the decomposition, isolate the intersections with $\mathcal{C}$.[6] Create vertices in the graph corresponding to these intersection points and to the critical points. For describing the arcs connecting these vertices in the graph, we assimilate, for simplicity, the points and the graph vertices. In each critical box the topology is simple: the critical point is connected to each of the intersection points on the boundary of the box.

There are several approaches to do the connections in the other rectangles of the decomposition. The usual and conceptually simplest is to refine boxes of extreme points so as to avoid top and bottom crossings; then, the number of left and right crossings in rectangles always match and the connection is one-to-one. Since we allow top/bottom crossings for efficiency (see above), this straightforward method does not apply. Another approach (see [AMW08, Sak91]) is to compute the sign of the slope of the tangent to the curve at the top/bottom crossings (this yields whether the top/bottom crossing should be connected to vertex to the right or to the left of its rectangle). We however want to avoid such additional computations.

For computing the connections in the non-critical rectangles of the decomposition, we use the multiplicities in fibers of the extreme points and a greedy algorithm. The geometric meaning of the parity of this multiplicity is the following: if it is even, the curve makes a U-turn at the extreme point, else it is odd and the curve is x-monotone in the neighborhood of the extreme point. Still, there are some difficulties for connecting the vertices, as illustrated on Figure 2(b): on the left is the information we may have on the crossings for two extreme points with x-overlapping boxes; the second and third drawings are two possible connections *in the middle rectangle* for different parities of the multiplicities. To distinguish between these two situations we compute the connections in rectangles starting from the top such that the connections in a rectangle below a critical box are computed once the connections in all the rectangles above the box are done.

The connections can easily be computed as follows. First, if there are vertical asymptotes, we have already determined in Step 5 whether a point that lies on the boundary of an asymptote box belongs to a branch that lies to the left or to the right of its asymptote; recall that such a point lies on a horizontal side of the box. Consider a rectangle R of the decomposition that is adjacent to an asymptote box, say below it (the case where it is above is similar); note that the top wall of R is a subset of the bottom wall of the asymptote box. The vertices on the top wall of R split into k_l vertices that are left of the asymptote and k_r vertices that are right of it. Each of the k_l vertices necessarily connects to a vertex on the left or bottom side of R. Moreover, among these k_l vertices, the i-th vertex starting from the left, connects to the i-th vertex on the left-bottom sides of R starting from the top. Similarly, among these k_r vertices, the i-th vertex starting from the right, connects to the i-th vertex on the right-bottom sides of R starting from the top (see Figure 2(a)).

Once these connections for asymptotes are done, due to requirements on extreme point boxes, there is at most one, not already connected, vertex on the *top or bottom* of any rectangle. The

[6] For simplicity, we ensure, thanks to refinements, that the curve never intersects an endpoint of an edge, that is, a corner of a rectangle. Note also that the intersections are already isolated on the sides of the critical boxes; an isolating interval may, however, need to be refined if it contains a vertex of the rectangle decomposition.

problem now is to determine if such a vertex should be connected to a vertex on the right or on the left side of the rectangle. The connections in the unbounded rectangles above critical boxes are straightforward: the ones between the vertices on the two vertical sides are in one-to-one correspondence, starting from infinity, and if a vertex remains on a vertical side, then there is a vertex on the horizontal side which it has to be connected with. Now, once all the connections have been computed in the rectangle(s) above the box of an extreme point, these connections and the multiplicity of the extreme point allows us to compute the connections in the rectangle(s) below, see Figure 2(b). Indeed, if there is a vertex on the bottom side of the critical box, then it lies on the top side of a rectangle. Inside this rectangle, the vertex is connected to the topmost vertex on the left or on the right side, depending on the multiplicity of the extreme point and on the side of the connection of the branch above the extreme point. The other connections in this rectangle and in the other rectangles below the critical box, if any, are performed similarly as for unbounded rectangles. Note that the two unbounded rectangles (the leftmost and the rightmost) that are vertical half-planes are treated separately: for each vertex on the vertical side we associate an arc that goes to infinity.

Output. A graph isotopic to the curve is output. In addition, x-extreme points, singular points and vertical asymptotes are identified and their position is approximated by boxes, hence refinable to any desired precision.

3.2 Curve with vertical lines

We assumed in the previous section that the curve $\mathcal{C}$ has no vertical line. In order to generalize the algorithm, we explain how to calculate the topology of a curve $\mathcal{C}_F$ defined by a rational bivariate polynomial $F(x, y) = 0$, which has vertical lines. The idea is first to process the curve without its vertical lines, then study the intersections of this curve with the vertical lines. Technically, these two processings are intertwined:

- The vertical lines have as x-coordinates the roots of V_l, the gcd of the coefficients of F seen as a polynomial in y. V_l is an univariate polynomial in x, its roots are isolated (vertical lines can be thought of as vertical *stripes*). The curve $\mathcal{C}_f$ with $f = \frac{F}{V_l}$ which has no vertical line, is processed as explained before, until Step 4 included.

- Intersections between $\mathcal{C}_f$ and the vertical lines generically occur on non-critical points of $\mathcal{C}_f$. Nevertheless, this may not always be the case and we need to identify critical points of $\mathcal{C}_f$ that also are on vertical lines. Solving the singular and extreme ideals of $\mathcal{C}_f$ with the additional polynomial V_l enable to identify which critical point of $\mathcal{C}_f$ is on which vertical line. Note that these points are singular for $\mathcal{C}_F$.

- Next additional boxes are created to witness the new relevant points of $\mathcal{C}_F$ and refinement is processed to meet the criteria of the connection step of the former algorithm. In more details, the x-intervals of critical points of $\mathcal{C}_f$ and vertical line stripes are refined until a vertical line stripe overlaps a critical box if and only if this critical point is on the line. We create new boxes, called *vertical*, that contain every point that is an intersection between a vertical line and $\mathcal{C}_f$ and that is not a critical point of $\mathcal{C}_f$. We refine the extreme point boxes of $\mathcal{C}_f$ and vertical boxes until there is at most one crossing with $\mathcal{C}_F$ on top (and bottom).

- Then Step 5 for vertical asymptotes of the previous algorithm is performed with the following modifications: For the computation of the bound M_x, we consider all vertical boxes in addition to critical boxes. We identify which vertical lines are also asymptotes by computing $gcd(V_l, V_a)$. We refine vertical lines stripes and asymptote boxes so that a vertical line stripe, whose line is not an asymptote, does not overlap any asymptote box. We add crossing points on asymptote boxes whose asymptote is also a vertical line.

- Finally, Step 6 for connection of the previous algorithm computes a graph isotopic to the curve $\mathcal{C}_F$.

3.3 Correctness of the algorithm

All algebraic computations are certified, hence the only thing that has to be proved is that the output graph $\mathcal{G}$ is isotopic to the input curve $\mathcal{C}$. Our proof is constructive and elementary, we define the ambient isotopy F as follows. On the skeleton of the rectangle decomposition F_t is chosen to be the identity for any t. It remains to define F on each rectangle. In a rectangle that is not a critical box, $\mathcal{C}$ is a set of x-monotone non-crossing arcs and $\mathcal{G}$ is a set of straight line segments connecting the same pairs of points on the rectangle's boundary, as $\mathcal{C}$ does. F_1 is defined on each section $x = \alpha$ by mapping the points of $\mathcal{C}$ to that of $\mathcal{G}$ so that their ordering on $x = \alpha$ is preserved, and by linear interpolation on the other points. F_t is then defined by linear interpolation $F_t = tId + (1-t)F_1$. To handle critical boxes, note that once cut vertically at the critical point, they behave like other rectangles (by Lemma 3).

3.4 Example

In this section, we show an example to illustrate the algorithm. We choose the curve $f(x, y) = y^4 - 6y^2x + x^2 - 4y^2x^2 + 24x^3$, as in [GVN02].

Before starting, one has to eliminate from the curve the vertical components. These vertical components are the gcd of the coefficients of f seen as a univariate polynomial in y. This gcd is 1: the curve has no vertical components and the algorithm will behave as in section 3.1.

Step 1 of the algorithm calculates boxes for the critical points of the curve f. First, the system of singular points I_s is solved.

$$\begin{aligned} I_s &= \mathbb{I}(f, f_y, f_x) \\ &= \mathbb{I}(y^4 - 6y^2x + x^2 - 4y^2x^2 + 24x^3, \quad 4y^3 - 12xy - 8yx^2, \quad -6y^2 + 2x - 8y^2x + 72x^2). \end{aligned}$$

The Gröbner basis of the system I_s with respect to the lexicographic ordering is calculated, giving $(3y^2 - x, xy, x^2)$. The RUR of this basis is the following; recall that the solutions of the system are $\left(x = \frac{G_1(t)}{G_0(t)}, y = \frac{G_2(t)}{G_0(t)}\right)$ for t solution of $F(t) = 0$.

$$F(t) = t^3, \quad G_0(t) = 3, \quad G_1(t) = 0, \quad G_2(t) = 3t.$$

As F has only one root, $t = 0$, the system I_s of singular points has only one solution $p_1 = (0, 0)$. Accordingly, our implementation reports that I_s has only one root in the box $[0, 0] \times [0, 0]$.

The system I_e of extreme points is given by $I_e = \mathbb{I}(f, f_y, 1 - u\, f_x)$; here the new variable u is added, ensuring that $f_x \neq 0$ for any solution of the system (indeed if $f_x = 0$ then $1 - u\, f_x = 1$ for

any u in $\mathbb{C}$). The system I_e is then

$$I_e = \mathbb{I}(y^4 - 6y^2x + x^2 - 4y^2x^2 + 24x^3, \quad 4y^3 - 12xy - 8yx^2, \quad 1 - u(-6y^2 + 2x - 8y^2x + 72x^2)).$$

To solve this system, we compute a Gröbner basis eliminating u and with lexicographical ordering on the other variables, giving $(72x^2 + 4 - 35y^2 + 99x, y^3 - 9xy + 4y, 3y^2x + 2 - 13y^2 + 48x)$. The associated RUR is

$$F(t) = t^5 - 19t^3 + 70t, \qquad G_0(t) = 1680 + 120t^4 - 1368t^2,$$
$$G_1(t) = -70 + 143t^4 - 1133t^2, \qquad G_2(t) = 912t^3 - 6720t.$$

F has five real roots, and each one of them maps to a root of the system I_e. The software reports (small) boxes containing each one of these extreme points $p_2, \ldots, p_6$. For clarity of the exposition, we consider enlarged versions of these critical boxes, as shown in Figure 3(a).

At this point, the isolating boxes of the extreme points are pairwise disjoint (and similarly for the singular boxes if there were more than one) but nothing ensures that the extreme boxes do not overlap with the singular boxes. The algorithm thus refine the boxes (by refining the isolating intervals of the roots of polynomials $F(t)$ in the RURs of I_e and I_s, and using interval arithmetic to obtain the refined 2D boxes) until all the boxes are pairwise disjoint. In this example, no refinement is needed because the boxes do not overlap.

The fact that the solver found exact coordinates for some points, here p_1, is actually a difficulty because, for instance, the number of intersection points of the curve with the boundary of a singular box does not yield the number of branches that are incident to the singular point. Point boxes are thus enlarged, initially to $[-\frac{1}{128}, \frac{1}{128}] \times [-\frac{1}{128}, \frac{1}{128}]$, and refined until they intersect no other critical boxes. This yields a box for p_1 which is $[-\frac{1}{4096}, \frac{1}{4096}] \times [-\frac{1}{4096}, \frac{1}{4096}]$.

In **Step 2**, the algorithm calculates the multiplicities of the critical points in fibers. For the singular point p_1, it calculates the smallest integer k such that the k^{th} derivative does not vanish. This is done by considering the systems $I_{s,k}$ obtained by adding f_{y^k} to the system $I_{s,k-1}$ with $I_{s,1} = I_s$ (for efficiency purpose, we actually add f_{y^k} to the Gröbner basis of $I_{s,k-1}$ which has already been computed when considering $I_{s,k}$).

Starting from $k = 2$, the solutions of $I_{s,k}$ are computed via a Gröbner and RUR calculations. Note that the solutions of $I_{s,k}$ are also solutions of $I_{s,k-1}$. The isolating boxes of the solutions of $I_{s,k-1}$ and $I_{s,k}$ are then refined until every box of each system intersects at most one box of the other system. This ensures that two intersecting boxes necessarily correspond to the same root of the two systems. We can thus easily decide whether a root of $I_{s,1}$ is also a root of $I_{s,2}, I_{s,3}, \ldots$ In our example the situation is quite simpler because $I_{s,1}$ has only one root, and thus any root of $I_{s,k}$ is necessarily that one. Still, computing the solutions of $I_{s,k}$ starting from $k = 2$, yields that the solution p_1 of $I_{s,1}$ is a solution of $I_{s,2}$ and $I_{s,3}$, but not of $I_{s,4}$. Hence, p_1 has multiplicity 4 in its fiber. Here, the systems $I_{s,k}$ and their Gröbner bases $Gb\_I_{s,k}$ are:

$$I_{s,1} = I_s \qquad Gb\_I_{s,1} = \{3y^2 - x,\ xy,\ x^2\}$$
$$I_{s,2} = I(3y^2 - x,\ xy,\ x^2,\ 12y^2 - 12x - 8x^2) \qquad Gb\_I_{s,2} = \{x,\ y^2\}$$
$$I_{s,3} = I(x,\ y^2,\ 24y) \qquad Gb\_I_{s,3} = \{y,\ x\}$$
$$I_{s,4} = I(y,\ x,\ 24) \qquad Gb\_I_{s,4} = \{1\}$$

On the other hand, the multiplicities of the five x-extreme points are computed using Teissier formula (see Lemma 2): the multiplicity of each point in its fiber is the multiplicity of the corresponding roots in the RUR of I_e plus one. In this case, $p_2, \ldots, p_6$ have multiplicity one in the RUR, which implies that they all have multiplicity 2 in their fibers.

Step 3 of the algorithm deals with the refinement of the boxes containing x-extreme points, that is, the boxes containing $p_2, \ldots, p_6$. The goal of this step is to obtain boxes such that the curve intersects every box's boundary at most twice, with at most one intersection on the top (resp. bottom) of the box. Each box is treated independently.

The intervals defining the box enclosing p_2 computed during the first step of the algorithm are $x_{p_2} = \left[\frac{4611611823926328587}{2305843009213693952}, \frac{461176103935218815}{2305843009213693952}\right]$ and $y_{p_2} = \left[-\frac{8627721659600529273}{2305843009213693952}, -\frac{8627594605412894855}{2305843009213693952}\right]$. The algorithm isolates the roots on the vertical walls, $f(x_{p_2,left}, y)$ and $f(x_{p_2,right}, y)$ in the interval y_{p_2}, and similarly for the horizontal walls. Two intersection points are found, one on the top wall, the other on the bottom wall. This means that the box containing p_2 does not need to be refined.

The situation is identical for the boxes containing p_3, p_5 and p_6. The box containing p_4 has also two intersections with the curve, here both on the left wall, which again does not require further refinement.

In **Step 4**, the algorithm refines the boxes enclosing singular points. The method uses the multiplicity k of each singular point (in this case, the only singular point is p_1) and the k^{th} derivative of the curve with respect to y (both were calculated in Step 2). Every singular point is treated independently; here with have only one.

We use interval arithmetic to ensure that f_{y^k} does not vanish in a box: computing with interval arithmetic, if the image of the box by f_{y^k} does not contain zero, then f_{y^k} does not vanish in the box (the converse is not true). We thus refine the box until its image by f_{y^k} does not contain zero.

We then refine the box in x until the curve $\mathcal{C}$ intersects neither the top nor the bottom of the box. Actually, before doing this last refinement, for efficiency purposes, we actually enlarge the box in y as much as we can, under the two constraints that it should continue to avoid the curve f_{y^k} and all the other critical boxes. The final resulting box of p_1 is $\left[-\frac{1}{4096}, \frac{1}{4096}\right] \times \left[-\frac{1}{4}, \frac{1}{4}\right]$ and it was not necessary to refine the box in x.

Step 5 deals with vertical asymptotes. Their x-coordinates are the roots of the leading coefficient of $f(x, y)$ view as a polynomial in y. The leading term of f is y^4, and the leading coefficient is 1, that is, the curve f has no vertical asymptotes.

Step 6 is the last stage of the algorithm, in which the graph isotopic to the input curve $\mathcal{C}$ is computed. The boxes containing critical points induce a subdivision of the plane in rectangles, as shown in Figure 3(b), and the intersection points between $\mathcal{C}$ and every wall of this subdivision are computed. As explained in Section 3.1, we connect vertices in a straightforward manner inside the critical boxes (see Figure 3(c)). In this example, the connections in the other rectangles of the subdivision are also straightforward (see Figure 3(d)) and we do not need to use the multiplicity of the extreme points nor to use a greedy approach. Figure 3(e) shows the same graph, using finer refinements, which is computed in 0.3 seconds on a standard PC.

Program output

The curve used as example of our algorithm is introduced as input of our MAPLE worksheet. The example is ran on a MacBook Pro, Intel Core 2 Duo, 2.6 GHz with 4Gb RAM. The printout is slightly modified for readability.

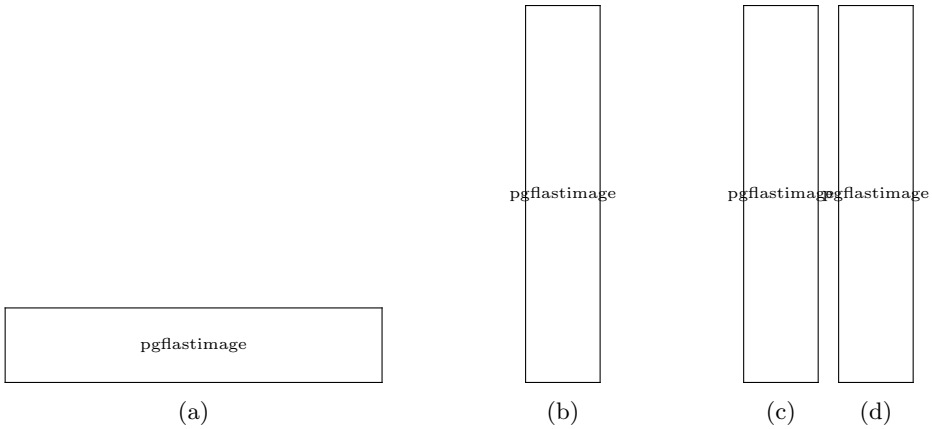


Figure 3: (a) Critical boxes of f. The small square at the origin contains the only singular point, p_1. Each other rectangles contains one of the x-extreme points $p_2, \ldots, p_6$. (b) Rectangular decomposition of the plane induced by the critical boxes. (c) Topology inside the critical boxes. (d) Graph isotopic to the curve. (e) Graph drawn with a finer refinement. (f) Detail of the graph near the origin.

```
# set the verbosity level:
verbose:=2:
# Optional but provides a smoother curve: set the precision for root isolations
# (2^{-precision}) and the number of additional vertical subdivisions of the
# non-critical rectangles of the decomposition.
precision:=10:  visualize:=true:  visualize_split:=10:

f := y^4 - 6*y^2*x + x^2 - 4*y^2*x^2 + 24*x^3:
isotop(f);
```

The program produces the following output with the graph shown in Figure 3(e); Figure 3(f) shows the details near the origin. The singular point is marked by a (red) diamond, and the extreme points are marked by (green) squares.

```
1. Compute boxes:
Vertical asymptotes (0 found) and vertical lines (0 found) computed in
  0.000000 seconds
Extreme Gröbner basis obtained in 0.013000 seconds
RUR calculated in 0.001000 seconds
Univariate isolation done in 0.004000 seconds
Singular Gröbner basis obtained in 0.015000 seconds
RUR calculated in 0.001000 seconds
Univariate isolation done in 0.004000 seconds
Computed 1 singular and 5 x-extreme points in 0.039000 seconds
Boxes of critical points refined to avoid overlap in 0.001000 seconds
Compute singular points of multiplicity k in the fiber for k=2
Compute singular points of multiplicity k in the fiber for k=3
Compute singular points of multiplicity k in the fiber for k=4
Multiplicities of singular points in fibers computed in 0.052000 seconds
Multiplicities of singular points in fibers are: [4]
Boxes of extreme points refined for topology in 0.032000 seconds
Boxes of singular points refined for topology in 0.003000 seconds
Total time for computing the boxes of critical points (1 singular and
  5 extreme) = 0.129000 seconds (including 0.091000 seconds for Gb/RS)

2. Sweep:
Partitionned the plane into 49 rectangles in 0.096000 seconds
Elapsed total computation time: 0.225000 seconds

3. Construct graph: done in 0.001000 seconds
Total computation time: 0.226000 seconds
```

4 Complexity Analysis

This section details the proof of the following theorem stating the bit complexity of our algorithm for the computation of the topology of a curve. We consider the Turing machine model of computation

and $\widetilde{\mathcal{O}}_B$ denotes the bit complexity where poly-logarithmic factors are omitted. Let an algebraic curve $\mathcal{C}$ be given by a square-free polynomial $f \in \mathbb{Z}[x, y]$ of total degree bounded by d and coefficients of bitsize bounded by τ. Let R be the number of critical points of the curve.

Theorem 4. *The bit complexity of our algorithm for the computation of the topology of the curve $\mathcal{C}$ is $\widetilde{\mathcal{O}}_B(R\, d^{22}\, \tau^2)$, which is $\widetilde{\mathcal{O}}_B(N^{26})$, where $N = \max\{d, \tau\}$.*

Definitions and notation. The bitsize of a rational number is defined as the maximum bitsize of its numerator and denominator. The bitsize of a polynomial is the maximum bitsize of its coefficients. In order to simplify notation, we assume in the sequel that $d = \mathcal{O}(\tau)$. However, we still express complexities in terms of d and τ when it is simple enough since in practice d may be much smaller than τ. We may also assume that the univariate polynomials that we compute with, are square free. This assumption does not change the complexity since the computation of their square-free part and computations with their square-free part, is of no extra cost. Indeed, for a polynomial of degree d and bitsize τ, its square-free part has degree $\mathcal{O}(d)$ and bitsize $\mathcal{O}(d+\tau) = \mathcal{O}(\tau)$ and it can be computed in $\widetilde{\mathcal{O}}_B(d^2\tau)$ [LR01]. We use the notion of separation bound of a polynomial (or of a zero-dimensional system of polynomial equations) which is the minimum distance between its (complex) roots. We call the bitsize of a separation bound s the minimum integer $\sigma \geqslant 0$ such that $s > 2^{-\sigma}$. For simplicity, we do not consider poly-logarithmic factors in the complexity bounds that we denote $\widetilde{\mathcal{O}}$.

Isolations of roots of systems via a RUR require some machinery from interval analysis, we briefly recall the basics and refer to [AH83] for additional details. For an interval $A = [a_1, a_2]$, let its width be $w(A) = a_2 - a_1$ and its absolute value be $|A| = \max(|a_1|, |a_2|)$. For a two-dimensional box $A \times B$, let $w(A \times B) = \max(w(A), w(B))$ and $|A \times B| = \max(|A|, |B|)$. We denote by $I(\mathbb{R}^n)$ the set of products of n real intervals. A polynomial has a natural extension to a function over $I(\mathbb{R}^n)$ by a process we call evaluation with interval arithmetic. More precisely, for a polynomial f, we denote by the bold letter $\mathbf{f}$ the corresponding interval function defined over $I(\mathbb{R}^n)$ by replacing the usual operations $+, -, \times$ by interval operations (note that the order in which the operations are processed can change the result, but this issue is irrelevant for our computations). In the sequel, we bound the number of times we refine the isolating intervals of the roots of the univariate polynomial of the RUR. We assume that every refinement divides by at least two the interval width.

Overview of the section. In order to derive the bit complexity of the algorithm presented in Section 3 we analyze the complexity of each step. The first two steps consists of computing the isolating boxes of the critical points and multiplicities. The analysis of these steps is presented in Section 4.2. In the third step, we refine the isolating boxes of the x-extreme points. The complexity of this step is bounded by the complexity of the next one, so we do not consider it explicitly. During the first part of the fourth step we refine the isolating boxes of the singular points with respect to their isolating curve. The analysis of this step is done in Section 4.3. During the second part, we refine the isolating boxes of the singular points, until the curve does not intersect them on the top and on the bottom. The analysis of this operation is done in Section 4.4. We conclude the proof of Theorem 4 in Section 4.5. Some technical details are postponed to Section 4.6.

4.1 Preliminaries

Before analyzing each step of the algorithm, we state the bitsize complexity of some recurrent basic computations. Let f be a univariate integer polynomial of degree d and bitsize τ, and x be a rational of bitsize σ.

Lemma 5 ([Yap00])**.** *The bitsize of the separation bound of f is in $\mathcal{O}(d\tau)$. Similarly, the bitsize of the endpoints of isolating intervals of the roots of f is in $\mathcal{O}(d\tau)$. Moreover, the absolute value of the roots of f is in $\mathcal{O}(2^\tau)$.*

We omit the proof of the following lemma which is straightforward.

Lemma 6. *The evaluation of f over x has complexity $\widetilde{\mathcal{O}}_B(d(\tau + d\sigma))$, while the number $f(x)$ has bitsize $\mathcal{O}(\tau + d\sigma)$.*

Since the interval evaluation of the operations $+, -, \times$ are a constant time more expansive that their usual counterparts we obtain the corollary:

Corollary 7. *The evaluation of f using interval arithmetic over an interval I with end points of the same bitsize as x, i.e. the computation of $\mathbf{f}(I)$, has complexity $\widetilde{\mathcal{O}}_B(d(\tau+d\sigma))$, while the interval $\mathbf{f}(I)$ has end points with bitsize $\mathcal{O}(\tau + d\sigma)$.*

We postpone to Section 4.6 the proof of the following lemma bounding the increase of the width of an interval by evaluation by interval arithmetic of a polynomial.

Lemma 8. *Let P be a univariate rational polynomial of degree d and bitsize τ, and A be an interval such that $|A| \leqslant 2^\sigma$ with $\sigma \geqslant 0$; then*

$$w(\mathbf{P}(A)) \leqslant 2^{\tau+d\sigma} d^2 w(A).$$

Let Q be a bivariate rational polynomial of total degree d and bitsize τ, and B be an interval such that $|B| \leqslant 2^\sigma$ with $\sigma \geqslant 0$; then

$$w(\mathbf{Q}(A, B)) \leqslant 2^{\tau+d\sigma+1} d^3 w(A \times B).$$

4.2 Computation of the isolating boxes

The first step of the algorithm is the computation of the isolating boxes of the singular and the extreme points. For the complexity of this step it suffices to compute the complexity of solving the system of critical points $I_c = \mathbb{I}(f, f_y)$.

The steps for solving the system are the followings. We compute the Gröbner basis of I_c and the RUR of the system. We solve the univariate polynomial of the RUR and, using the isolating intervals of its real roots, we compute boxes that contain the real solutions of the systems. Finally, we refine the boxes until they are all disjoint.

Gröbner basis and RUR. We compute the Gröbner basis of I_c with the degree reverse lexicographic order in $\widetilde{\mathcal{O}}_B(d^8\tau)$ [Laz83]. Next we compute the RUR representation of the solution of the system and multiplicities in $\widetilde{\mathcal{O}}_B(d^{13}\tau)$, the reader may refer to [Rou99] for more details. Besides I_c we have to solve the systems $I_{s,k}$, for $1 \leqslant k \leqslant d$, to determine the multiplicities of the singular points. We may assume that each of them could be solved in $\widetilde{\mathcal{O}}_B(d^{13}\tau)$, as in the case of I_c, since the polynomials have degrees bounded by d and the bitsize of the coefficient is bounded by $\widetilde{\mathcal{O}}(\tau)$. Hence the total complexity is $\widetilde{\mathcal{O}}_B(d^{14}\tau)$. However, this is an overestimation since the systems are over-determined and thus the Gröbner bases could be computed faster [BFS04] and in practice we don't need to solve all d of them.

The RUR representation has the following form: $h(T) = 0, x = \frac{g_x(T)}{g_0(T)}, y = \frac{g_y(T)}{g_0(T)}$, where $h, g_0, g_x, g_y \in \mathbb{Q}[T]$. The polynomial $h(T)$ is actually the so called *u-resultant* [Can88]. It has degree $\mathcal{O}(d^2)$ and bitsize $\mathcal{O}(d^2 + d\tau) = \mathcal{O}(d\tau)$. One way to see this is to consider a curve (and thus the system I_c) in generic position, this is without loss of generality since shearing the curve in generic position does not increase the bitsizes more than by a factor in $\mathcal{O}(\lg(d))$. In this case we can consider as the polynomial $h(T)$ the projection of the system on the x-axis or, in other words, the resultant of the system with respect to y. Under this notion, g_0 and g_x, respectively g_0 and g_y could be seen as the coefficients of the first non-vanishing sub-resultant of f and f_y, with respect to y, respectively with respect to x. Thus, the degree of these polynomials is $\mathcal{O}(d^2)$ and their bitsize is $\mathcal{O}(d\tau)$.

We can simplify the expressions of x and y, if we take into account that $h(T)$ and $g_0(T)$ are relative prime. Indeed, we can compute a polynomial $g(T)$ such that $g \cdot g_0 = 1 \mod h$ and thus express the coordinates of the solutions as $x = h_x(T) = g_x(T)\, g(T)$ and $y = h_y(T) = g_y(T)\, g(T)$. The degree of g is $\mathcal{O}(d^2)$ and its bitsize is $\mathcal{O}(d^3\tau)$, as g is a Bézout coefficient of the extended Euclidean division of g_0 and h [vzGG03]. The same bounds hold for $h_x(T)$ and $h_y(T)$.

Computation of the boxes. In order to solve the system, it suffices to compute the isolating intervals of the real roots of h, and substitute them in h_x and h_y using interval arithmetic. This gives isolating boxes that contain the real roots of the system. Finally, we refine these boxes until they become disjoint.

Lemma 9. *To ensure that the boxes of the critical points are disjoint, it is sufficient to refine $\mathcal{O}(d^3\tau)$ times each isolating interval of the corresponding root of the univariate polynomial of the RUR of the critical points.*

Proof. One needs to refine the isolating intervals of the real roots of h until the corresponding boxes of the system, computed by interval evaluation with h_x and h_y, become disjoint. In other words, the isolating boxes should have width smaller than 2^{-s_c}, where s_c is the separation bound of the system of critical points. If $2^{-\mu}$ is a lower bound on the width of the isolating intervals of h, Lemma 8 yields a value μ, so that the evaluations by h_x and h_y give intervals of width at most 2^{-s_c}. We consider only the polynomial h_x, since the computation is similar for h_y. Lemma 8 applied with h_x of bitsize $\mathcal{O}(d^3\tau)$ and degree $\mathcal{O}(d^2)$ evaluated at the roots of h of absolute value $\mathcal{O}(2^{d\tau})$ (by Lemma 5) yields:

$$2^{\mathcal{O}(d^3\tau)+\mathcal{O}(d^2)\mathcal{O}(d\tau)} d^2 2^{-\mu} \leqslant 2^{-s_c}.$$

Thus, it suffices to consider μ in $\mathcal{O}(s_c + d^3\tau)$.

On the other hand, the separation bound of the critical points is larger than the separation bounds of the x (or y) coordinates of these points. The coordinates are roots of the resultant of f and f_y with respect to y (or x), which is a polynomial of degree $\mathcal{O}(d^2)$ and bitsize $\mathcal{O}(d\tau)$. Hence the separation bound of the coordinates is of bitsize $\mathcal{O}(d^3\tau)$ (by Lemma 5). This is also a bound for the separation bound of the critical points thus s_c is in $\mathcal{O}(d^3\tau)$ and $\mathcal{O}(s_c + d^3\tau)$ is also in $\mathcal{O}(d^3\tau)$.

By Lemma 5, the isolating intervals of h have initial width $\mathcal{O}(2^{d\tau})$. Thus, to reduce them to the width $2^{-\mu}$, it suffices to perform $\mathcal{O}(d\tau + d^3\tau) = \mathcal{O}(d^3\tau)$ refinements, provided that the refinements divide the interval widths by at least two. □

4.3 Refinement with respect to the isolating curve

We require that the isolating boxes of singular points avoid their associated curve $f_k = \partial^k f/\partial y^k$. This is ensured by refining the isolating boxes of the singular points so that the evaluation of f_k, using interval arithmetic, results an interval that does not contain zero.

Lemma 10. *To ensure that the boxes of singular points do not overlap their associated curve f_k, it is sufficient to refine $\widetilde{\mathcal{O}}(d^4\tau)$ times the corresponding roots of the univariate polynomial of the RUR.*

Proof. Consider a box $B = I \times J$ that isolates a singular point p. Assume, without loss of generality, that the width of B is $2^{-\mu}$. Let k be such that f_k is the isolating curve for p. We need to ensure that the evaluation of f_k over B does not contain 0. A sufficient condition is that $w(\mathbf{f_k}(B)) < \delta$ with $\delta < |f_k(p)|$. Defining s_v as the maximum bitsize of the values of f_k at the singular points of f (more precisely $s_v = \max_{s\in\mathbb{N}^*}\{|f_k(p)| > 2^{-s}, p \text{ singular point of } f \text{ and } f_k(p) \neq 0\}$), we can choose $\delta = 2^{-s_v}$.

f_k is of degree $\mathcal{O}(d)$ and bitsize $\widetilde{\mathcal{O}}(d\tau)$ (by Stirling formula). The absolute value of a box of a singular point is $\mathcal{O}(2^{d\tau})$ since the x or y coordinates of a box are roots of the resultant of f and f_y with respect to y or x, and such a resultant has bitsize $\mathcal{O}(d\tau)$ (Lemma 5). Lemma 8 applied for the evaluation of f_k over a singular point box yields:

$$2^{\widetilde{\mathcal{O}}(d\tau)+\mathcal{O}(d)\mathcal{O}(d\tau)+1} d^3 2^{-\mu} \leqslant 2^{-s_v}.$$

Thus, it suffices to consider μ in $\mathcal{O}(s_v + d^2\tau)$.

On the other hand, Lemma 13 gives that s_v is in $\widetilde{\mathcal{O}}(d^4\tau)$, hence μ is in $\widetilde{\mathcal{O}}(d^4\tau)$.

Let $2^{-\nu}$ be the width of the isolating interval of h that corresponds to the singular point via the RUR. Then ν should be such that the box computed via the RUR has width $2^{-\mu}$. Applying Lemma 8 again, for the evaluation of h_x of degree $\mathcal{O}(d^2)$ and bitsize $\mathcal{O}(d^3\tau)$ over this isolating interval of absolute value $\mathcal{O}(2^{d\tau})$, ν should satisfy:

$$2^{\mathcal{O}(d^3\tau)+\mathcal{O}(d^2)\mathcal{O}(d\tau)} d^2 2^{-\nu} \leqslant 2^{-\mu}.$$

Thus, it suffices to consider ν in $\widetilde{\mathcal{O}}(d^4\tau)$.

We conclude, as in the proof of Lemma 9, that it suffices to perform $\widetilde{\mathcal{O}}(d^4\tau)$ refinements. □

4.4 Refinement of the singular points to avoid top/bottom crossings

The last step that we need to consider is the analysis of the refinement of the isolating boxes of the singular points, until there is no intersection between the curve and their top and bottom sides.

Lemma 11. *To avoid top/bottom intersection between the curve and the boxes of singular points, it is sufficient to refine $\widetilde{\mathcal{O}}(d^9\tau)$ times the corresponding roots of the univariate polynomial of the RUR.*

Proof. Consider a singular point and its isolating box refined according to Lemmas 9 and 10. We further refine the x-coordinate of the box until the top (or equivalently the bottom) side does not intersect the curve. The line supporting the top side is of the form $y = c$, for some constant c. Hence, the x-coordinates of the intersections of the curve with the top side are among the roots of $f(x, c)$. Consider the polynomial P whose roots are the roots of $f(x, c)$ and the x-coordinates of

the critical points. A sufficient condition to avoid intersection on the top side is to ensure that the x-width of the box is smaller than the separating bound of P.

The bitsize of c is the same as that of the evaluation of h_x over an end-point a of an isolating interval of a root of h. From Lemmas 9 and 10, the bitsize of a is in $\widetilde{\mathcal{O}}(d^4\tau)$. Since h_x is of degree $\mathcal{O}(d^2)$ and bitsize $\mathcal{O}(d^3\tau)$, c has bitsize $\widetilde{\mathcal{O}}(d^6\tau)$ (Lemma 6). Hence $f(x,c)$ is a polynomial of degree $\mathcal{O}(d)$ and bitsize $\widetilde{\mathcal{O}}(d^7\tau)$. The polynomial P is the product of $f(x,c)$ and the resultant with respect to y of f and f_y, thus its degree is in $\mathcal{O}(d^2)$ and its bitsize is in $\widetilde{\mathcal{O}}(d^7\tau)$. The bitsize of the separation bound of P is thus δ in $\widetilde{\mathcal{O}}(d^9\tau)$

Let $2^{-\mu}$ be the width of the isolating intervals of h corresponding to the singular point by the RUR. Lemma 8 applied with h_x of bitsize $\mathcal{O}(d^3\tau)$ and degree $\mathcal{O}(d^2)$ yields:

$$2^{\mathcal{O}(d^3\tau)+\mathcal{O}(d^2)\mathcal{O}(d\tau)}d^2 2^{-\mu} \leqslant 2^{-\delta}$$

Thus, it suffices to consider μ in $\widetilde{\mathcal{O}}(d^9\tau)$.

We conclude, as in the proof of Lemma 9, that it suffices to perform $\widetilde{\mathcal{O}}(d^9\tau)$ refinements.

□

4.5 Overall complexity

Combining the results from Section 4.2, 4.3 and 4.4 we can now prove Theorem 4.

Proof of Theorem 4. Combining the results of Lemmas 9, 10, and 11, we prove that the algorithm performs $\widetilde{\mathcal{O}}(d^9\tau)$ refinements. Each refinement consists of an evaluation of h, h_x and h_y over a rational number of bitsize $\widetilde{\mathcal{O}}(d^9\tau)$. Using Horner's rule, each evaluation of these polynomials of degree $\mathcal{O}(d^2)$ and bitsize $\mathcal{O}(d^3\tau)$ has complexity $\widetilde{\mathcal{O}}_B(d^2(d^3\tau+d^2d^9\tau)) = \widetilde{\mathcal{O}}_B(d^{13}\tau)$ (Lemma 6). The complexity of the $\widetilde{\mathcal{O}}(d^9\tau)$ refinements is thus in $\widetilde{\mathcal{O}}_B(d^{13}\tau d^9\tau) = \widetilde{\mathcal{O}}_B(d^{22}\tau^2)$. If there are R singular points the total cost is thus $\widetilde{\mathcal{O}}_B(Rd^{22}\tau^2)$.

Note that the costs of Gröbner and RUR computations are dominated. Finally, the complexity of dealing with vertical asymptotes (Step 5), vertical lines and the connection part of the algorithm (Step 6) is dominated by the complexity of the other steps.

Finally, if $N = \max\{d,\tau\}$, note that R is in $\mathcal{O}(d^2) = \mathcal{O}(N^2)$, and so the total complexity of the algorithm is $\widetilde{\mathcal{O}}_B(N^{26})$. □

4.6 Technical details

4.6.1 Interval arithmetic

In our algorithm, we need, several times, to evaluate univariate and bivariate polynomials over intervals. This is done using classical interval arithmetic operations. Theoretically, we need to control how large an interval becomes when a polynomial operation is performed. We prove here Lemma 8, which we recall for convenience.

> *Let P be a univariate rational polynomial of degree d and bitsize τ, and A be an interval such that $|A| \leqslant 2^\sigma$ with $\sigma \geqslant 0$; then $w(\mathbf{P}(A)) \leqslant 2^{\tau+d\sigma}d^2 w(A)$.*
>
> *Let Q be a bivariate rational polynomial of total degree d and bitsize τ, and B be an interval such that $|B| \leqslant 2^\sigma$ with $\sigma \geqslant 0$; then $w(\mathbf{Q}(A,B)) \leqslant 2^{\tau+d\sigma+1}d^3 \max(w(A), w(B))$.*

Proof of Lemma 8. We apply the basic formulas for the sum and the product of intervals [AH83, Theorem 9, p.15], which are for any real number a and integer $n \geqslant 1$:

$$\begin{array}{ll} w(A \pm B) = w(A) + w(B), & w(aA) = |a| w(A), \\ w(AB) \leqslant w(A)|B| + |A| w(B), & w(A^n) \leqslant n|A|^{n-1} w(A). \end{array}$$

Let $P(x) = \sum_{i=0}^{d} c_i x^i$ with $|c_i| \leqslant 2^\tau$ and $Q(x,y) = \sum_{i,j \geqslant 0}^{i+j \leqslant d} c_{ij} x^i y^j$ with $|c_{ij}| \leqslant 2^\tau$. We have:

$$\begin{aligned} w(\mathbf{P}(A)) &= \sum_{i=0}^{d} |c_i| w(A^i) \quad \leqslant \quad 2^\tau \sum_{i=0}^{d} i|A|^{i-1} w(A) \quad \leqslant \quad 2^\tau w(A) d \sum_{i=1}^{d} |A|^{i-1} \\ &\leqslant 2^\tau w(A) d^2 \max(1, |A|^{d-1}) \quad \leqslant \quad 2^\tau w(A) d^2 2^{d\sigma} \quad \leqslant \quad 2^{\tau + d\sigma} d^2 w(A). \\ w(\mathbf{Q}(A,B)) &= \sum |c_{ij}| w(A^i B^j) \quad \leqslant \quad 2^\tau \sum w(A^i)|B|^j + w(B^j)|A|^i \\ &\leqslant 2^\tau \sum i w(A)|A|^{i-1}|B|^j + j w(B)|A|^i |B|^{j-1} \\ &\leqslant 2^\tau d^3 2 w(A \times B) \max(1, 2^{\sigma(d-1)}) \quad \leqslant \quad 2^{\tau + d\sigma + 1} d^3 w(A \times B). \end{aligned}$$

□

4.6.2 Separation bounds

In this section, we compute the separation bound needed for the proof of Lemma 10. We first need a refinement, due to Yap [Yap00], of Gap theorem [Can88].

Theorem 12 ([Yap00] Gap theorem 11.45)**.** *Let $\Sigma = \{A_1, \ldots, A_n\} \subseteq \mathbb{Z}[x_1, \ldots, x_n]$ be a system of n polynomials, not necessarily homogeneous. Suppose that Σ has finitely many complex zeros and $(\xi_1, \ldots, \xi_n)$ is one of these zeros. Assume $d_i = \deg(A_i)$ and*

$$K := \max\{\sqrt{n+1}, \max\{\|A_i\|_2 \,|\, 1 \leqslant i \leqslant n\}\},$$

where $\|A_i\|_2$ is the usual Euclidean norm of the vector of coefficients of the polynomial A_i. If $|\xi_i| \neq 0$, $i = 1, \ldots, n$, then

$$|\xi_i| > (2^{2/3} N K)^{-D}\, 2^{-(n+1) d_1 \cdots d_n},$$

where

$$N := \binom{1 + \sum_{i=1}^{n} d_i}{n}, \quad D := \left(1 + \sum_{i=1}^{n} \frac{1}{d_i}\right) \prod_{i=1}^{n} d_i.$$

We now prove, following closely [Yap06], the lemma we use in the proof of Lemma 10.

Lemma 13. *Let $\mathbf{p}$ be a critical point of a curve $\mathcal{C}_f$, without vertical lines, defined by a square-free polynomial f of degree d and bitsize τ. Let $f_k = \partial^k f / \partial y^k$, where $2 \leqslant k \leqslant d$. If $\mathbf{p}$ is not a point on f_k, then $|f_k(\mathbf{p})| > 2^{-s_v}$ with s_v in $\widetilde{\mathcal{O}}(d^4 \tau)$.*

Proof. We consider the following system for $k \in \{2, \ldots, d\}$

$$\begin{array}{lcl} A_1 & : & f(\mathbf{p}) = 0, \\ A_2 & : & f_y(\mathbf{p}) = 0, \\ A_3 & : & h - f_k(\mathbf{p}) = 0. \end{array}$$

The system is zero-dimensional because the number of critical point is finite, and we can apply Theorem 12, where

$$d_1 = d, \quad d_2 = d-1, \quad d_3 \leqslant 2(d-k),$$

$$N \leqslant \binom{4d-2k}{3} \leqslant 16d^3, \quad D \leqslant (d+2)(d-1)(2d-1) \leqslant 4d^3.$$

The bitsize of the norm of a polynomial is the bitsize if the polynomial itself. Since the bitsize of f and f_y is in $\mathcal{O}(\tau)$ and that of f_k is in $\widetilde{\mathcal{O}}(d\tau)$, we conclude that the bitsize of K is in $\widetilde{\mathcal{O}}(d\tau)$. The factor that gives the bitsize in the lower bound of Theorem 12 is thus K^{-D} and h is bounded by a value of bitsize in $\widetilde{\mathcal{O}}(d^4\tau)$. □

5 Experiments

We implemented our algorithm, ISOTOP, in MAPLE using the Gb/RS MAPLE package [FGb, RS], implemented in C, for computing Gröbner bases, RURs and isolating roots. We believe that comparing MAPLE and C/C++ implementations is fair for our problem when the running time is not too small because then, most of the time is usually spent on algebraic computations which are coded in C/C++ (possibly in the kernel of MAPLE). When the running time is too small, the MAPLE part of the code is not negligible and comparing MAPLE and C/C++ implementations becomes meaningless. This is why we focused our tests on examples for which the running time exceeds 1 second. We measure the running time for computing the isotopic graph, but not the drawing. All the experiments were performed using 2.6 GHz single-core Pentium 4 with 1.5Gb of RAM and 512kb of cache, running 32-bit linux.

We compared our code, ISOTOP, with two C++ implementations Alcix [EKW07] and Cad2d [Bro02] and two MAPLE implementations, Top [GVN02] and Insulate [SW05]. Another promising software is Axel, [AMW08] but no implementation of the certified subdivision algorithm is currently available.

Alcix is a C++ code, part of the CGAL library [CGA].[7] Cad2d is a stand-alone C++ code which can also be compiled in combination with the Singular library [GPS01] (used for polynomial factorization). In our tests, Cad2d appears to be much more efficient when ran with Singular (and we report these tests). Finally, recall that Top requires an initial precision, which we set to 50.

As discussed in Section 1, the various implementations do not compute exactly the same thing and comparisons should thus be taken with care. Recall that when the curve is not in generic position, Top and Insulate do not compute the critical points (and, in particular, the x-extreme points) in the original coordinate system. ISOTOP, Alcix and Cad2d always output the critical points in the original coordinate system.

We ran large scale benchmarks on over a thousand of curves during several weeks. In particular, we considered curves suggested in [Lab08, Bro02, GVN02] and several classes of non-generic curves.[8] We considered about 1300 curves from [Lab08], which are classified in 18 challenges covering a large variety of interesting cases such as isolated points, high multiplicity of tangency at singularities, large number of branches at singularities or many singularities. This set contains curves of degree

[7] Following the recommendation of M. Kerber, we ran two versions of the code with the flag CGAL_ACK_RESULTANT_FIRST_ STRATEGY set to 1 and 0. One being optimized for generic cases, while the other is optimized for singular curves. We always compare to the better running time.

[8] The logs are available at `http://vegas.loria.fr/isotop/benchmarks`.

up to 90 that are both in generic and non-generic position. As suggested in [Bro02], typical curves in generic position can be generated (i) as a random bivariate polynomial (which usually do not have singular points) or (ii) as resultants of two random trivariate polynomials (which usually have singular points, including isolated points). In both cases, we considered random polynomials with 50% non-zero coefficients of bitsize 32 in Case (i) and initial bitsize 8 in Case (ii). We generated such curves with degrees up to 25. We also generated classes of curves in non-generic position in two different ways. First, we considered products of a curve with one or several of its vertical translates. Second, we considered curves of the type $g = f^2(x, y) + f^2(x, -y)$; such curves are usually irreducible and consist of isolated points which are the intersections of the curve C_f with its symmetric with respect to the x-axis. We generated such curves with degrees up to 24.

We set in our experiments a limit of 30 minutes for the computation of the topology of one curve. We report as *time out* instances that exceed this running time. Also, Cad2d which uses Singular for modular arithmetic, often reports on difficult instances that the table of primes has been exhausted, which results in an interruption of computation; this is reported in the tables as *aborted*.

In summary, we ran our benchmarks on a total of 1500 curves. As mentioned above, it is not significant to compare C++ and MAPLE implementations when the running time is too small. We thus only report experiments on 650 curves whose running times exceeded 1 second for ISOTOP. The distribution of degrees and number of critical points of these 650 curves is shown in Figure 4.

Figure 5 shows the ratio of running times between each of the competing implementations and ISOTOP over our set of 650 curves. It appears difficult to analyze the benchmarks globally because there are always particular examples that are processed faster by a given implementation. We note, however, that Insulate is almost always slower than ISOTOP, except for random curves with no singular points. In addition, Insulate and Top reached the time limit on more than half of the examples and, in particular, on difficult examples. We can, nevertheless, comment on the general behavior of the different approaches depending on the classes of examples.

To illustrate the behavior on curves in generic positions, we report the running times for random curves in Table 1 and for resultants of surfaces in Table 2. Random curves have no singular points and few extreme points. In this case, we observe that ISOTOP is the least efficient implementation. This can be explained by the fact that ISOTOP computes the Gröbner basis of a large system without multiplicities, which is the worst case in practice. On the other hand, the other implementations benefit from interval arithmetic filters in the lifting phase, which speed up computations by avoiding expensive symbolic computations, see for example [Bro02]. Generic curves generated as resultants have many singularities and extreme points. ISOTOP benefits from splitting the critical system in two smaller (singular and extreme) systems and hence it performs relatively better than in the completely random case. We observe that ISOTOP is typically a bit slower than Alcix but faster than Top, and that Cad2d aborts.

To illustrate the behavior on curves in non-generic position, we consider different classes of curves. The first class of non-generic curves are constructed with one curve multiplied by one or several of its vertical translates. The initial curve is taken either randomly, in Table 3, or it is a resultant of two surfaces, in Table 4. Table 5 reports results on the second class of non-generic curves of the type $f^2(x, y) + f^2(x, -y)$ for random polynomials f. For these non-generic curves, ISOTOP is typically faster than other implementations.

As a general rule, we observe that, except for random curves, that is, curves in generic position and without singular point, the ratio of the running times between other implementations and

ISOTOP is increasing with the degree of the curve. In other words, except for random curves, ISOTOP tends to perform better, compared to others, when the degree increases.

6 Conclusion

We presented a new algorithm and implementation for computing the topology of plane algebraic curves. Instead of a CAD (cylindrical algebraic decomposition) based method, our algorithm relies upon Gröbner bases, Rational Univariate Representations and hence avoids computations with algebraic numbers even in non-generic cases. A strength of our approach is to be insensitive to the non-genericity of the curve. As demonstrated by the experiments, our implementation is competitive with state-of-the-art C/C++ implementations in the case of generic curves and faster for high-degree non-generic ones. Future work includes taking advantage of the possible decomposition of the curve into factors. We already decompose the system of critical points into systems of extreme and singular points. One natural step further would be to consider the primary decompositions of the ideals and thus work with systems of lower complexity.

Acknowledgments

We would like to thank the authors of Alcix, Cad2d, Insulate, and Top for supplying their code and O. Labs for his list of curve challenges. ET is partially supported by an individual postdoctoral grant from the Danish Agency for Science, Technology and Innovation.