Solving bivariate algebraic systems and topology of plane curves

Yacine Bouzidi

To cite this version:

Yacine Bouzidi. Solving bivariate algebraic systems and topology of plane curves. Symbolic Computation [cs.SC]. Université de Lorraine, 2014. English. <NNT : 2014LORR0016>. <tel-00979707>

HAL Id: tel-00979707

https://tel.archives-ouvertes.fr/tel-00979707

Submitted on 16 Apr 2014

École doctoral IAEM Lorraine, Département de formation doctorale en informatique

---

THÈSE DE DOCTORAT

Pour obtenir le titre de

Docteur de l'université de Lorraine

Spécialité : INFORMATIQUE

Présentée par

Yacine BOUZIDI

Résolution de systèmes bivariés et topologie de courbes planes

Soutenue le 18 mars 2014 devant un jury composé de :

Rapporteurs :	Laurent BUSÉ (INRIA Sophia Antipolis-Méditerranée) Laureano GONZALEZ-VEGA (Universidad de Cantabria) Eric SCHOST (University of Western Ontario)
Examinateurs :	Sylvain LAZARD -directeur- (INRIA Nancy Grand Est - Loria) Marc POUGET -co-directeur- (INRIA Nancy Grand Est - Loria) Fabrice ROUILLIER (INRIA Paris Roquencourt) Michael SAGRALOFF (Max-Planck-Institut für Informatik Saarbrücken) Monique TEILLAUD (INRIA Sophia Antipolis - Méditerranée)

Laboratoire Lorrain de Recherche en Informatique et ses Applications - UMR 7503

Remerciements

Je voudrais, en premier lieu, exprimer ma reconnaissance la plus sincère à Sylvain Lazard et Marc Pouget, pour avoir encadré ma thèse et pour m'avoir accompagné avec autant de rigueur, de sérieux et de disponibilité tout au long de ces trois années d'effort. Travailler à leurs côtés a été un réel plaisir et m'a beaucoup apporté à la fois sur le plan scientifique que humain.

Je voudrais également remercier profondément Fabrice Rouillier qui m'a donné goût à la discipline du calcul formel, d'abord à travers ces cours passionnants en master, puis durant le stage de fin d'études effectué à ses cotés. Son enthousiasme, ses nombreuses idées et son amour sans limite pour la recherche scientifique sont autant de qualités qui ont fait du travail avec lui une véritable fête. Trugarez !

Je tiens à remercier chaleureusement Eric Schost, Laurent Busé et Laureano Gonzalez-Vega d'avoir accepté de rapporter ma thèse et Michael Sagraloff et Monique Teillaud d'avoir accepté de faire partie de mon jury. Leurs nombreuses remarques et conseils sont la preuve de l'intérêt qu'ils ont porté à mon travail.

Je voudrais également remercier Guillaume Moroz avec qui j'ai eu plaisir à discuter, et qui s'est toujours montré disponible pour répondre à mes nombreuses questions. Ses "très très" nombreux pointeurs bibliographiques, ses commentaires ainsi que ses pertinentes remarques m'ont été d'une grande utilité dans l'aboutissement de mes travaux.

Je remercie aussi tous mes collègues du Loria. Ceux d'abord de l'équipe VEGAS qui m'ont si gentiment accueillit et ont veillé à ce que je me sente bien au sein du groupe, Sylvain Petitjean, Xavier Goaoc, Laurent Dupont, Luis penaranda et Guillaume Batog. Ceux ensuite, que j'ai côtoyé à l'intérieur des murs du laboratoire ou bien en dehors, et avec qui, j'ai eu des échanges passionnants, drôles et parfois vifs pendant le déjeuner ou lors des innombrables pauses café. Une spéciale dédicace au cheikh Nazim Fatès, mon collègue et ami qui porte le tarbouche et chante l'andalou comme personne.

A tous mes amis avec qui j'ai passé de formidables moments à discuter de sujets, il est vrai prosaiques, mais au combien drôles, à jouer au football, à me changer les idées en somme, je dis également merci. Merci en particulier à Mehdi d'avoir toléré ma grivoiserie, et à Ahmed mes talents de footballeur, merci aussi à Rafik, Karim, Younes, Judith, Imene, Adrien, Lionel, Soumeya, Alexandre, Lilia, et à Feriel mon fidèle souvenir ...

Merci également au groupe de parigots, chez qui j'allais respirer lorsque je manquais d'air à Nancy. Khaled la Mannschaft, Rabah la scoumoune et mourad, à la fois pourfendeur et jouisseur du libéralisme. Merci enfin, à tout ceux que je pourrais avoir oublié.

Pour terminer, j'ai une pensée particulière à toute ma famille, sans qui, tout cela n'aurait jamais été possible. Votre soutient indéfectible et votre confiance m'ont aidé à braver la tourmente et m'ont permit d'arriver à bon port saint et sauf. A Vous donc, papouni, mamouni, les 3K (Kechrouda, Kahlouch et Krimo) et le nouveau venu Akacha, je vous aime !

A ma mère, je dédis ces quelques vers d'Alfred de Musset

Ô toi, dont les soins prévoyants,
Dirigent mes pas nonchalants,
Des écueils d'un monde trompeur
Je veux devoir tout mon bonheur

Dans les sentiers de cette vie
Ma mère, à toi je me confie.
Écarte ma faible nacelle.
A la tendresse maternelle.

Contents

Chapter 1

Introduction

Contents

1.1 The Problem

Computational geometry is the field of computer science dedicated to the establishment of theoretical foundations for the study of geometric algorithms. Traditionally, the focus has been on linear objects such as points, line segments, polygons and polyhedra. The "simple" structure of these objects eases the design of algorithms dealing with them as well as their analysis. In real world however, most problems related to geometry are naturally modeled by curved objects and a non-trivial task consists in providing algorithms that compute efficiently and accurately with these objects. This explains why, recently, tools that come from other fields such as algebra and topology, have been used by the computational geometry community in order to improve the accuracy and the efficiency of these algorithms. Typically solving algebraic systems plays an increasing role in computational geometry algorithms that deal with non-linear objects. Indeed, as non-linear objects are often defined as a zero set of polynomials, computing the roots of polynomial systems yields information about the structure and the properties of these objects. Among the recent work that uses algebraic system solving in computational geometry, one can mention for example the work of Everett et al. [Everett 2007] where several systems are solved in order to prove a conjecture on the Voronoi diagrams of three lines in general position in three-dimensional space.

In this context, the computational geometry problem we will focus on in this thesis is the computation of the topology of a real algebraic plane curve. We consider such a curve $\mathcal{C}$, defined in a Cartesian coordinate system by a bivariate polynomial f with rational coefficients i.e. $\mathcal{C} = \{(x, y) \in \mathbb{R}^2 | f(x, y) = 0\}$ with f in $\mathbb{Q}[x, y]$. This problem is fundamental since it is central in the problem of plotting the curve in a certified way, that is in a way where the obtained plot is isotopic to the curve (see below). Last years, much work has been dedicated to the problem of computing the topology of curves leading to the development of several algorithms (see Section 1.3 for details).

In vague terms, computing the topology of a curve $\mathcal{C}$ consists in computing a geometric representation that describes completely the structure of $\mathcal{C}$. This representation usually consists of a planar graph $\mathcal{G}_c$ whose vertices are mapped to points of the curve and such that connecting these points by segments yields a drawing isotopic to the input curve. In other words, $\mathcal{C}$ can be transformed continuously into $\mathcal{G}_c$ without topological changes, see Figure 1.1.

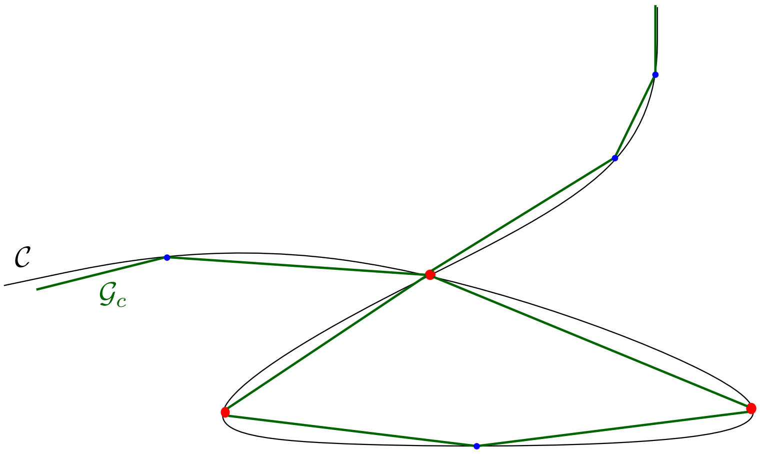

Figure 1.1: $\mathcal{G}_c$ is an isotopic graph associated to the curve $\mathcal{C}$.

In order to obtain an isotopic graph, the graph 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 usually mapped to vertices of the graph. In addition, in order to preserve the geometry of $\mathcal{C}$, the computed graph usually contains vertices that correspond to the extreme points of $\mathcal{C}$ for a particular direction (say the direction of the x-axis), that is, points that are non-singular and for which the tangent line is vertical (i.e., parallel to the y-axis). These singular and extreme points form the critical points of $\mathcal{C}$, and an important step in the construction of an isotopic graph of $\mathcal{C}$ is the determination of these points. Since these points are defined as the solutions of some bivariate systems that consists of f and some of its partial derivatives, computing efficiently these critical points requires to efficiently solve systems of bivariate polynomials.

It should be stressed upfront that the problem of solving polynomial systems is one of the main topics in computer algebra and many successful theoretical and practical results have been obtained. However, most of these algorithms treat the case of general polynomial systems and very few address specifically the case we are interested in, that is the case of systems with bivariate polynomials with rational coefficients. This explains why, an interesting challenge consists in investigating the case of bivariate systems.

In the present document we are going to study, elaborate and implement efficient and robust algorithms for solving systems of bivariate polynomials with coefficients in $\mathbb{Q}$. Beside the fact that this problem is fundamental in our problem of computing the topology of curves, solving bivariate systems also arises in many other application fields as for example robotic, computer aided design or molecular biology.

Before going further, it is important to answer the following questions:

- What do we mean by solving systems?
- What do we mean by robustness?
- How do we evaluate the efficiency?

It is not an easy task to provide clear and definitive answers to these questions. The answers often depend on the context of the computation and the application fields for which the algorithms are designed. However, having as objective the problem of computing the topology of plane curves, we try to provide clear answers to these questions.

1.1.1 Solving bivariate algebraic systems

In abstract terms, solving a polynomial system means computing all its solutions. Although simple, this definition is difficult to translate in computer terms where the notion of “solution of a polynomial system” is not clearly defined. In some contexts, solving a system amounts to computing some formal representation of its solutions that allows to deduce more or less easily information on these solutions such as numerical approximations, number of solutions, etc. Such a representation can be for example a Gröbner basis, a rational parameterization or a triangular representation. In some other contexts, the emphasis is put on the computation of numerical approximations of the solutions. These numerical approximations are usually given as a set of isolating axis-parallel boxes such that every real solution of the system lies in a unique box and conversely. Isolating boxes for the solutions can be computed directly from the input system using numerical approximation methods such as Newton method, or they can be obtained from a formal representation of the solutions. It should be stressed however that formal solutions do not necessarily yield, directly, isolating boxes of the solutions. In particular, from a theoretical complexity view, it is not proved that the knowledge of a triangular system or Gröbner basis of a system always simplifies the isolation of its solutions. This observation also explains why it is not an easy task to define precisely what a formal solution of a system is, and why usage prevails in what is usually considered to be a formal solution.

In this thesis, we choose to represent the solutions of bivariate systems using a particular type of rational parameterization called Rational Univariate Representation [Rouillier 1999]. Such a representation is given by a univariate polynomial f and two rational functions $(\frac{f_x}{f_1}, \frac{f_y}{f_1})$ that define a one-to-one mapping between the roots of f and the solutions of the system (see Figure 1.2 for an illustration).

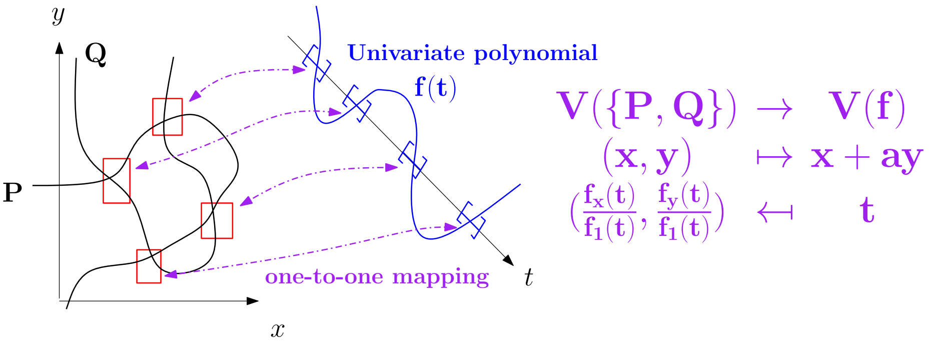
y
Q
P
x
Univariate polynomial
$\mathbf{f(t)}$
t
one-to-one mapping
$\mathbf{V(\{P,Q\}) \rightarrow V(f)}$
$\mathbf{(x,y) \mapsto x+ay}$
$\mathbf{(\frac{f_x(t)}{f_1(t)}, \frac{f_y(t)}{f_1(t)}) \leftarrow t}$

Figure 1.2: Rational Univariate Representation of the system $\{P, Q\}$ where $V(S)$ is the set of (complex) solutions of a system S of polynomials.

All the univariate polynomials of the Rational Univariate Representation, i.e. $\{f, f_1, f_x, f_y\}$, are uniquely defined with respect to a *separating linear form*, that is, a linear combination of the variables $x + ay$ that takes different values when evaluated at different solutions of the system. Also, computing a Rational Univariate Representation requires to first compute such a linear form which, at present, turns out to be the dominant part in the computation of this representation. This difficulty explains why in practice, based on the fact that a random linear form is separating with high probability, most of the algorithms that compute rational parameterizations proceed by choosing randomly a linear form and then computing the associated mapping, which results in probabilistic Monte-Carlo algorithms for computing such a representation. In some other algorithms, the chosen linear form is tested for separation to eliminate the Monte-Carlo aspect but the problem of deciding whether a linear form is separating turns out to be a difficult task.

In addition to the fact that a Rational Univariate Representation is conceptually simple, the choice of such a representation can be motivated in several respects as discussed below.

Starting from a Rational Univariate Representation, one can easily compute isolating boxes for the real solutions of the system by first computing isolating intervals of the real roots of f (using any univariate isolation algorithm, see for instance [Rouillier 2003, Mehlhorn 2008]) and then computing the image of these intervals by the rational fractions $\frac{f_x}{f_1}$ and $\frac{f_y}{f_1}$. This is important in our problem of computing the topology since isolating boxes around the critical points of the curve are needed as an input of the connection step. An algorithm for efficiently computing isolating boxes of the solutions from the associated Rational Univariate Representation is discussed in detail in Chapter 6.

Another notable advantage of Rational Univariate Representations is that they enable the transformation of problems involving solutions of multivariate systems into problems on roots of univariate polynomials. One relevant example (discussed in detail in Chapter 6) is the famous Sign_at operation that consists in computing the sign of an arbitrary bivariate polynomial at one solution of a bivariate system.

Finally, another important property of Rational Univariate Representations is that they give information about the multiplicity of the solutions. This information is useful in many applications and especially for computing the topology of curves where some multiplicities are often needed to determine the topology inside the boxes containing critical points of the curve (see Chapter 6 for details).

1.1.2 Robustness and probabilistic algorithms

In this thesis, the emphasis is put on complete and certified algorithms for solving bivariate systems and computing the topology of plane curves. In other words we aim for methods that always return the correct result for all types of input. For this purpose, three aspects need to be taken into account, the robustness, completeness and probabilistic aspects. By robustness we mean robustness in the context of numerical rounding errors in a fixed-precision floating-point arithmetic. By completeness we refer to the ability of an algorithm to handle any input, possibly in degenerate form. Finally by probabilistic aspects, we refer especially to algorithms that do, or do not, return a correct result, and distinguish in particular between Monte-Carlo and Las-Vegas algorithms. These aspects are discussed in more details below.

Robustness. The question of exact computation is critical, especially in computational geometry algorithms. Indeed, the correctness of these algorithms is generally proved under the assumption that some basic geometric predicates are computed exactly whatever is the input. In the context of algebraic input as we consider here, the exact representation of algebraic numbers would require an infinite precision in floating point representation. Hence, we use instead the classical implicit representation via isolating intervals. This consists in representing the solutions of algebraic systems by isolating intervals with multiple-precision bounds and in providing algorithms for computing exactly with these solutions in, for instance, sign computations, comparison, etc (see [Yap 2000, §6.3] for the case of roots of univariate polynomials).

Completeness. Our objective is the design of general algorithms that do not assume any genericity conditions on the input. The precise meaning of genericity varies depending on the problem and the algorithm. For instance, in the case of topology computation, a very common condition is that the input curve is in generic position, that is, no two critical points are vertically aligned. In the case of system solving, a typical genericity condition is that the ideal corresponding to the input system is radical, that is, no multiple solutions exist. This is the case for example with numerical methods based on Newton iteration where the correctness as well as the termination are guaranteed under the condition that no multiple solutions exist. We however allow algorithms that make genericity assumption when this assumption is verified during the computation. For instance, when it comes to solving bivariate systems we always assume in our algorithms that the system has finitely many solutions and we provide a way to check that it is indeed the case.

Probabilistic algorithms. Many algorithms in computer algebra depend on a generic choice. This choice is most of the time random in practice which makes these algorithms probabilistic. These probabilistic algorithms are generally of two types: Monte-Carlo algorithms whose running time is bounded, but whose output may be incorrect with a certain (typically small) probability, and Las-Vegas algorithms who take an amount of time that varies randomly and is not necessarily bounded, but always produce the correct result.

A school-book example that illustrates such algorithms is the famous coin toss problem, where a coin is successively tossed aiming at obtaining a head (or tail). A natural Monte-Carlo algorithm consists in tossing a coin a constant number of times, say k. The running time of such an algorithm is then bounded by the time required to perform k tosses, while the probability to obtain the desired outcome depends on k. One can alternatively opt for the Las-Vegas variant of this algorithm that consists in tossing a coin until the first head outcome (or tail). In that case the desired outcome is guaranteed and only the running time is random, although not bounded.

A typical example of probabilistic Monte-Carlo algorithms in computer algebra are those computing rational parameterizations of the solutions of an algebraic system. As mentioned previously, all these algorithms require the computation of a separating linear form. In practice, a linear form is chosen randomly and the associated univariate representation is computed. However, although such a random choice is valid with high probability, it may happen that the chosen linear form does not separate the solutions and thus, that the computed univariate representation does not correctly encode the solutions of the system. Another example worth mentioning, is the methods that use computations over $\mathbb{Z}/\mu\mathbb{Z}$ combined with the Chinese Remainder Algorithm. This strategy is quite commonly used in symbolic algorithms to avoid intermediate growth of coefficients and thus speedup the computations (see Section 2.5 for details). The choice of the prime numbers μ is done randomly and it may happen that some of them are not appropriate for applying the Chinese Remainder Theorem which leads to a wrong result. Although in practice, such "unlucky" primes are rare, it is still possible to find examples for which, the algorithm returns an incorrect result.

Such Monte-Carlo algorithms can be transformed into Las-Vegas algorithms whenever there exists a way to verify that the output produced by the algorithm is indeed correct. If so then the resulting Las-Vegas algorithm is merely to repeatedly run the Monte-Carlo algorithm until one of the runs produces an output that is checked to be correct. It should be stressed however that often, testing the output for correctness is far from being trivial and induces a gap, from theoretical and practical point of view, between probabilistic Monte-Carlo algorithms and the corresponding Las-Vegas variants. A challenge that we will address in our work is to reduce as much as possible this gap in order to obtain efficient algorithms while keeping the Las-Vegas certification.

Note finally that, in some cases, the number of times that a probabilistic algorithm can fail, can be upper-bounded which, thus, also bounds the running time of the algorithm. This is for example the case with the algorithms presented in this thesis, where the number of bad choices (the "unlucky" prime numbers or the non-separating linear forms) is bounded and so as for the running time. Such algorithms are known under the name of randomized algorithms, and differ from the usual Las-Vegas algorithm where the running time is usually not assumed to be bounded. However, especially in computer algebra, one often uses the term Las-Vegas to refer to such algorithms because they have the typical structure of Las-Vegas algorithms, which is to repeat iteratively one procedure until it succeeds. We use the same terminology for the algorithms presented below.

1.1.3 Complexity and efficiency

We measure the efficiency of our algorithms both from the theoretical and practical point of view.

For the theoretical efficiency, we proceed in general by computing worst-case asymptotic complexities, that is the asymptotic number of operations performed by the algorithms. These complexities are bounded using the classical big O notation. Most complexity analyses in computer science measure the complexity in terms of arithmetic operations. However, especially in computer algebra, such complexities often underestimate the actual running time of the algorithm. The reason behind this, is the growth of coefficients during the computation, which increases the cost of arithmetic operations, thus affecting the global efficiency of the algorithm. In the case of polynomial with integer coefficients as we consider, a more relevant measure is the bit complexity, that is, the number of bit operations performed by the algorithm. In the context of algorithms on polynomials, such a complexity depends in general on two parameters: the total degree of the input polynomials and the maximum bitsize of their coefficients.

The bit complexity is hence a more precise measure of algorithms efficiency, but its computation is in general tedious since it requires a more in depth analysis of the algorithms than for the arithmetic complexity.

In addition, worst-case complexities are suitable to measure the intrinsic difficulty of the problem at hand, but the latter should however be taken with care when it comes to practical efficiency where the average analysis might be more relevant. This is particularly the case with randomized algorithms whose worst-case complexity may be very bad compared to the actual behavior of the algorithm. For such algorithm we compute upper bounds on the expected number of bit operations which we refer to as the expected bit complexity.

Finally, from the practical efficiency point of view, we compare the running time of our implementations with several state-of-the-art implementations on a large number of instances.

1.2 Contributions

In this thesis, we present several results, both theoretical and practical concerning the resolution of bivariate algebraic systems. These results mostly concern the computation of separating linear forms and Rational Univariate Representations. We also present contributions on strongly related problems such as the computation of isolating boxes for the solutions of such systems and the evaluation of the sign of other polynomials at the solutions of the systems. Finally, we present results on the initial problem that motivated our study, namely the computation of topology of plane algebraic curves.

In a nutshell, our main results are several algorithms for computing a RUR and isolating boxes of the real solutions of a system of two bivariate polynomials. The bit complexities of these algorithms are respectively $\widetilde{O}_B(d^8+d^7\tau)$ in the worst-case and, in average, in $\widetilde{O}_B(d^6+d^5\tau)$ in a Las-Vegas setting and also (but independently) in a Monte-Carlo setting, where d is the total degree of the input polynomials and τ is the maximum bitsize of their coefficients and where O_B refers to an asymptotic upper bound on the bit complexity and $\widetilde{O}$ means that polylogarithmic factors are omitted. Furthermore, some of these probabilistic algorithms (both Monte-Carlo and Las-Vegas) are shown to be very efficient in practice. We detail below a brief overview of our contributions.

Separating linear form. Our first contribution, which is the topic of Chapter 3, is a new deterministic algorithm of worst-case bit complexity $\widetilde{O}_B(d^8+d^7\tau)$ for computing a separating linear form of a system of two bivariate polynomials of total degree at most d and integer coefficients of bitsize at most τ. The system should be zero dimensional but this is tested in our algorithm. This decreases by a factor d^2 the best known complexity for this problem.

As a direct consequence, using our algorithm for computing a separating linear form directly yields a rational parameterization of the solutions of a bivariate system within the same overall complexity as our algorithm, both in the approach of Gonzalez-Vega et al. [Gonzalez-Vega 1996, Diochnos 2009] and in the one presented in Chapter 4 for computing a Rational Univariate Representation as defined in [Rouillier 1999]. As a byproduct, we obtain an algorithm for computing the number of (complex) distinct solutions of such systems within the same complexity, i.e. $\widetilde{O}_B(d^8+d^7\tau)$.

Rational Univariate Representations. In Chapter 4 we show that the Rational Univariate Representation (RUR for short) of Rouillier [Rouillier 1999] (i) can be expressed with simple polynomial formulas, that (ii) it has a total bitsize that is asymptotically smaller than that of Gonzalez-Vega and El Kahoui by a factor d, and that (iii) it can be computed with the same complexity, that is $\widetilde{O}_B(d^7 + d^6\tau)$. Namely, we prove that the RUR consists of four polynomials of degree at most d^2 and bitsize $\widetilde{O}(d^2 + d\tau)$ (instead of $O(d)$ polynomials with the same asymptotic degree and bitsize for the Gonzalez-Vega and El Kahoui parameterization). Moreover, we prove that this bound holds for the RUR of any ideal containing P and Q, that is, for instance the RUR of the radical ideal of $\langle P, Q\rangle$.

Efficient practical algorithms. Another aspect of our work is to investigate efficient practical algorithms for computing a decomposition of a bivariate system $\{P, Q\}$ into a set of Rational Univariate Representations. We present an approach that is similar to that in [Gonzalez-Vega 1996, Diochnos 2009] and show how the use of a multi-modular approach based on the Chinese Remainder Algorithm (see Section 2.5) combined with a random choice of a separating linear form can speed up the computation of a RUR from theoretical and practical point of view. Our first result is a Monte-Carlo algorithm with an expected bit complexity in $\widetilde{O}_B(d^6 + d^5\tau)$ and a good probability of success (whose parameter appears in the bit complexity in a logarithmic factor and is thus hidden). We then derive from this algorithm two Las-Vegas variants of expected bit complexity in $\widetilde{O}_B(d^7+d^6\tau)$ and $\widetilde{O}_B(d^6+d^5\tau)$ respectively. These Las-Vegas algorithms use two different methods to check that the result computed by the Monte-Carlo algorithm is correct. While the first method is intuitive, conceptually elegant and implemented, the second one shows that it is possible to design a Las-Vegas algorithm with the same asymptotic bit complexity as the Monte-Carlo algorithm.

Isolation of the real solutions. As our main objective is to obtain numerical approximations of the real solutions, we show in Chapter 6 that, given a RUR, isolating boxes of the solutions of the system can be computed with $\widetilde{O}_B(d^6 + d^5\tau)$ bit operations. This decreases by a factor d^4 the best known complexity for isolating the solutions of a bivariate system from a corresponding rational parameterization [Diochnos 2009]. Together with the previous results, this first brings the overall bit complexity of isolating the real solutions of a bivariate system via a univariate representation to $\widetilde{O}_B(d^8 + d^7\tau)$ in the worst-case. Note that this complexity equals the best known one proved by Emeliyanenko and Sagraloff [Emeliyanenko 2011] for solving bivariate systems using an alternative resultant-based algorithm (see related work below). This also brings the overall average bit complexity of isolating the real solutions via a univariate representation to $\widetilde{O}_B(d^6+d^5\tau)$ both in Monte-Carlo and Las-Vegas settings.

Applications. We finally exhibit some advantages of computing a RUR of a bivariate system through three applications: We first show how a rational univariate representation (and more generally a rational parameterization) can be used to perform efficiently two important operations on the input system. We first show how a RUR can be used to perform efficiently the *sign_at* operation. Given a polynomial F of total degree at most d with integer coefficients of bitsize at most τ, we show that the sign of F at one real solution of the system can be computed in $\widetilde{O}_B(d^8 + d^7\tau)$ bit operations, while the complexity of computing its sign at all the $\Theta(d^2)$ solutions of the system is only $O(d)$ times that for one real solution. This improves the best known complexities of $\widetilde{O}_B(d^{10}+d^9\tau)$ and $\widetilde{O}_B(d^{12}+d^{11}\tau)$ for these respective problems (see [Diochnos 2009, Theorem 14 & Corollary 24] with the improvement of [Sagraloff 2012] for the root isolation). Similar to the *sign_at* operation, we show that a RUR can be split in two parameterizations such that F vanishes at all the solutions of one of them and at none of the other. We also show that these two parameterizations can be transformed into RURs, in order to reduce their total bitsize, within the same complexity, that is, $\widetilde{O}_B(d^8 + d^7\tau)$.

Coming back to our initial problem, that is, the computation of the topology of curves, we show that solving bivariate systems through Rational Univariate Representations brings improvements in several respects to the algorithm presented in [Cheng 2010].

1.3 Related work

We discuss in the following, existing methods for isolating the real roots of systems defined by two bivariate polynomials with integer coefficients. We compare between the different state-of-the-art algorithms and give when provided the complexity of each of them. We further review different algorithms for computing the topology of a planar algebraic curve. The majority of these algorithms make extensive use of bivariate system solving techniques.

1.3.1 Bivariate systems solving

There exist many methods for isolating the real roots of system defined by two bivariate polynomials with integer coefficients. These methods can be classified in two different families. The family of numerical methods and that of symbolic methods. Note that we only briefly discuss the former category of methods since most of these methods are able to solve systems only under genericity conditions.

Numerical methods. Such methods treat the polynomials as real value functions and analyze the zero-level of these functions. They usually lead to local computations where the solutions are sought in a given area of interest, such as the famous Newton (-Raphson) method (see for e.g.

[Rheinboldt 1998]). The strength of numerical methods is undoubtedly their practical efficiency due essentially to the use of approximate computation. However, without convenient assumptions on the input system or additional algebraic computations, such methods seldom give guarantee of correctness and termination, especially in case of systems with singular solutions (solutions with multiplicities) where the Newton-like techniques are not applicable anymore. One important representative of this set of methods is the subdivision-based methods [Mourrain 2009, Garloff 2000, Sherbrooke 1993]. Subdivision can be seen as a generalization of the binary search, it starts with a box of interest and recursively splits it into smaller boxes, eliminating boxes which cannot contain a zero of the system, and ending up with a union of boxes that contains all solutions of the system that lie within the given box. In [Moore 2009], [Neumaier 1990, §5.1] and references therein, the check for the existence as well as the uniqueness of some solution inside a given box is achieved using numerical predicates based on interval computation techniques such as interval evaluation, Newton interval method, Krawzcyk operator [Rump 1983], etc.

Another set of numerical methods that are worth mentioning are those based on homotopy continuation (see e.g. [Li 2003, Verschelde 2010]). Roughly speaking, such methods use homotopy combined with numerical tracking algorithms to approximate the solutions of a polynomial systems.

Symbolic methods. These methods comprise the classical elimination-based methods such as resultant, triangular decomposition or rational univariate representation-based methods. Compared to the aforementioned purely numerical methods, symbolic methods are in general complete in the sense that no assumption is made on the system under consideration. In addition, the result, computed in a certified way provides global information on the set of the solutions (including the complex ones). These methods, which can be viewed as generalizations of the classical Gauss elimination method proceed generally in two steps: a symbolic step (also called a projection step), where a formal representation of the solutions is computed from the initial system using algebraic properties and polynomial combinations, and a numerical step (also called lifting step) that starts from this formal representation and computes numerical approximations of the solutions of the system. The reliance of such methods on costly symbolic computations (during the symbolic step) makes those methods slow and impractical for general polynomial systems as soon as their degree or number of variables become high. However they still constitute a suitable choice for solving systems of bivariate polynomials. Algebraic methods are of different sorts depending on the type of representation computed in the symbolic step. We discuss in the following three main types of representations, the resultant representations, the triangular representations and the univariate representations.

Resultant representation. These methods follow the same basic idea. They first project the solutions along several directions to derive a set of candidate solutions and then identify among these candidates those that are actually solutions. Note that for these methods, the projection step consists only of resultant computations. We survey below several significant results using this representation.

In [Seidel 2005] this projection idea is used to compute the set of critical points of an algebraic planar curve f, that is the set of the real roots of the system $\{f, \frac{\partial f}{\partial y}\}$. Under the assumption that the given curve is in generic position, i.e. there are no two critical points on the same vertical line, the algorithm projects the set of critical points, K, in three different directions. First onto the x- and the y-axis which yields a set $A \times B \supset K$ of candidate solutions given as pairs of isolating intervals, and then onto a third direction that is verified to be non-degenerate in order to recover the set of actual solutions K for $A \times B$.

In [Cheng 2009] a local generic position-based method called LGP is presented for solving bivariate systems. Roughly speaking, this method performs a coordinate transformation of the form $x \mapsto x + ay$ that transforms the initial system S into a new system S' that is in generic position, i.e. where different solutions are projected to different points on the x-axis. Then, the solutions of S are deduced as linear combinations of the projections onto the x-axis of the solutions of S and those of S'. The key contribution of this paper is the simple way the rational a in the linear form $x + ay$ is computed. However, one important drawback of this approach is that the rational a computed this way can be of bitsize, that of the separation bound of the resultant of the two input polynomials, i.e. $\widetilde{O}(d^3\tau)$. This has a negative impact on the cost of symbolic computations after performing the coordinate transformation. Note that for the two previous methods, no complexity analyses of the global solving algorithms are provided.

Unlike the two previous methods, the Grid method in [Diochnos 2009] does not assume, nor compute a generic position for the initial system. Instead, the algorithm projects the solutions onto the x- and the y-axes which yields a set of pairs of algebraic numbers and then selects among this set, the pairs at which the input polynomials vanish. Testing a pair is done by computing the sign of the input polynomials at the corresponding algebraic numbers, following the approach of Roy and Likteig [Lickteig 2001]. Although very simple to describe, this method suffers from the symbolic cost of the exact implementation of the Sign_at operation. The complexity analysis of this method shows that the overall bit complexity is dominated by that of the Sign_at operation which is in $\widetilde{O}_B(d^{14} + d^{13}\tau)$.

Recently, following the Grid approach in [Diochnos 2009], Berberich et al. [Berberich 2011b] proposed a very efficient algorithm for isolating the real roots of a bivariate system named Bisolve. The first step of this algorithm is the same as that in the Grid method (i.e. the solutions are projected onto the orthogonal axes by resultant computations). The second step is the separation, where isolating discs that separate the (complex) projected solutions from each others are computed using interval refinement [Kerber 2009b]. Finally, in a last step, an original inclusion predicate is used to validate a candidate solution as an actual real solution. This inclusion predicate based on interval arithmetic and Hadamard's bound on the determinant ensures, under some conditions on the size of the intervals, the existence of a real solution inside a given box. The complexity analysis provided in [Emeliyanenko 2012] shows that isolating the real solutions of systems of two bivariate polynomials using Bisolve can be done using $\widetilde{O}_B(d^8 + d^7\tau)$ bit operations where d and τ denote the degree and the bitsize of the input polynomials. It should be stressed that this complexity is the best known complexity so far for the problem of isolating the real solutions of a bivariate polynomial system.[1] Furthermore, the authors provide a very efficient implementation that benefits from a recent approach [Emeliyanenko 2010] to compute resultants exploiting the power of Graphics Processing Units (GPUs). The use of GPU computations reduces substantially the cost of the symbolic step which is known to be the bottleneck in such elimination-based algorithms. One drawback of this approach is, however, the portability of the software.

Triangular representation. Another way to express symbolically the solutions of a polynomial system is to use a triangular representation of the solutions. In the bivariate case, a triangular representation of a system has the form $\{U(x), B(x, y)\}$. The advantage of such a triangular representation with respect to the aforementioned resultant representation is that it eases, although arguably, the numerical approximations of the solutions, since the latter amounts to the isolation of the roots of univariate polynomials obtained successively by substituting the variables by the roots of the preceding polynomials.

The methods that compute such a triangular representation proceed generally by decomposing the initial system into a set of triangular systems such that the union of the solutions of these systems is exactly the solutions of

---

[1] One of the main contribution of this thesis, is an algorithm that achieves the same complexity bound, but by computing, in addition, a univariate representation of the solutions.

the initial system [Aubry 1999a, Lazard 1992].[2] For general systems of m polynomials with n variables, several algorithms exist for computing such a decomposition. Some of these algorithms proceed by variable elimination, that is, by reducing the solving of a system in n variables to that of a system in $n-1$ variables [Kalkbrener 1993, Wang 2001] while others proceed incrementally, that is, by reducing the solving of a system in m equations to that of a system in $m-1$ equations [Lazard 1991, Chen 2011]. However, few complexity results are known about the size and the computation of triangular representations of general polynomial systems. We can mention [Dahan 2004], where triangular representations of zero-dimensional varieties over the rational field are considered and polynomial bounds on the bitsize of their coefficients in terms of intrinsic quantities are proven.

Compared to general polynomial systems, several complexity results have been obtained for the case of systems defined by two bivariate polynomials with rational coefficients. In particular, Gonzalez-Vega and El Kahoui [Gonzalez-Vega 1996], present a simple algorithm based on polynomial subresultants, that computes a triangular decomposition of a bivariate system[3]. The bit-complexity of this algorithm analysed in [Diochnos 2009] for systems of two polynomials with degree d and bitsize τ shows an overall cost in $\widetilde{O}_B(d^7+d^6\tau)$ bit operations. In addition, an efficient implementation of this algorithm based on evaluation/interpolation techniques and computations over prime fields is provided in [Li 2011]. Recently, Lebreton et al. [Lebreton 2013] presented a probabilistic Monte-Carlo algorithm for computing a triangular decomposition of the set of non-singular solutions of a bivariate system. This algorithm relies on a combination of lifting techniques and fast modular composition. For the case of systems with integer coefficients, this algorithm was shown to have an essentially optimal bit complexity.

As mentioned above, the isolation of the solutions of a triangular system amounts to successively isolate polynomials with real algebraic numbers as coefficients. This is in fact non-trivial, especially when these polynomials have multiple roots. In [Boulier 2009], the authors propose an algorithm which assumes that the considered triangular system is squarefree. This algorithm uses a generalization of Vincent-Collins-Akritas (or Descartes) algorithm to isolate the real roots of polynomials with real algebraic number as coefficients. The algorithm in [Cheng 2007] makes no squarefreeness assumption on the triangular system and uses interval arithmetic combined with the so-called *sleeve* polynomials to isolate the real solutions. More precisely, it replaces the coefficients of the algebraic polynomial by sufficiently refined intervals, hence

---

[2] In the literature, the term *regular chain* is also used to refer to such triangular systems.

[3] This algorithm is described in details in Chapter 3 where it is used to compute the number of distinct solutions of a bivariate system.

obtaining upper and lower bounds (i.e. sleeve) for the polynomials. The isolation is performed using evaluation and exclusion predicates that involve the test to zero of the derivative. For the two previous algorithms, no complexity results are given but only implementations.

Recently, Strzebonski and Tsigaridas [Strzebonski 2012] analyze two different approaches for isolating the real roots of a univariate squarefree polynomial with coefficients in a simple algebraic extension. These approaches, applied to the problem of isolating the real roots of a triangular system yield a Las-Vegas algorithm whose expected bit complexity is $\widetilde{O}_B(d^8 + d^7\tau + d^6\tau^2)$.

Univariate representation. Another widespread representation is the univariate representation of the solutions. Recall that a univariate representation is a set of univariate polynomials and associated one-to-one mappings that send the roots of the univariate polynomials to the solutions of the system. The univariate representation of the solutions has a long history that can be traced back to the work of Kronecker at the end of the 19th century. In the last decades, such a representation has been extensively used under different forms for the study of zero-dimensional polynomial systems [Renegar 1989, Canny 1988, Giusti 2001, Alonso 1996, Rouillier 1999].

A univariate representation is defined with respect to a separating polynomial [Rouillier 1999] that is a polynomial that is injective on the set of the solutions i.e. that takes different values when evaluated at distinct (complex) solutions of the system. Also, the computation of any univariate representation usually decomposes in two different steps. A first step consists in computing a separating polynomial for the solutions and a second step consists in computing the polynomials defining the univariate representation. It should be stressed however that, among the algorithms that compute univariate representations, seldom search deterministically for a separating polynomial for the solutions. In general a polynomial is randomly chosen as a linear combination of the variables and the corresponding univariate representation is computed with the hope that the chosen polynomial is separating. Although such a random choice is good with high probability, without any further verification, this leads to probabilistic Monte-Carlo algorithms for the computation of univariate representations.

There exist plenty of algorithms for computing univariate representations. We can mention the work in [Giusti 2001, Durvye 2008] and references therein, where this representation is computed by means of the so-called geometric resolution. These algorithms assume that the ideal representing the set of solutions is radical since they use Hensel lifting which relies on Newton's iteration. Another way to compute such a representation is to use the concept of u-resultant [Canny 1988].

In [Rouillier 1999] the case of non-radical ideal is considered and an efficient algorithm is proposed for computing a Rational Univariate Representation of the solutions or RUR. The RUR is proven to be unique (up to a separating form) and preserves information on the multiplicities of the solutions. The algorithm in [Rouillier 1999], which is recalled in Chapter 5 for the case of bivariate systems, proceeds by performing linear algebra operations in the quotient algebra corresponding to the ideal. It requires the knowledge of the multiplication table of this quotient algebra, which can be computed for example from a Gröbner basis of the ideal.

In the case of systems of bivariate polynomials with integer coefficients, Gonzalez-Vega and El Kahoui [Gonzalez-Vega 1996] present an algorithm for computing a subresultant-based RUR that proceeds by computing a triangular decomposition of the input system. More precisely, the algorithm first applies a generic linear change of variables to the two input polynomials and then computes a decomposition into a set of rational representations using the subresultant sequence of the sheared polynomials. The complexity analysis of this algorithm in [Diochnos 2009] shows that the overall bit complexity is dominated by the deterministic computation of the separating form used to perform the generic change of variables that is $\widetilde{O}_B(d^{10} + d^9\tau)$. Once a linear separating form is computed, the polynomials of the univariate representations are obtained for the cost of a triangular decomposition, which is shown to be in $\widetilde{O}_B(d^7 + d^6\tau)$.

The main advantage of the univariate representation of the solutions with respect to the two previous forms of representations (the resultant and the triangular representation) is that it transforms the problem of isolating the real solutions of a system into the problem of first, isolating a *univariate* polynomial and, second, evaluating the image of the resulting intervals through the mappings of the rational representation. A naive analysis of this numerical step yields a bound in $\widetilde{O}_B(d^{10}\tau^2)$ bit operations, (see in [Cheng 2010], Lemma 9 and the proof of Theorem 4). However, as we will see in Chapter 6, this complexity can be improved.

1.3.2 Topology computation

The problem of computing the topology of plane algebraic curves is relatively recent compared to the system solving problem. The first papers dealing with topology computation can be traced back to the nineties. Since then, computing the topology of curves has become an active area of research and a fair number of results have been obtained on the subject (see e.g. [Burr 2008, Gonzalez-Vega 2002, Seidel 2005, Berberich 2011a] and references therein). Modern efficient algorithms for computing the topology of plane algebraic curves are of two types. One class of algorithms is based on subdivision techniques, typically, a quad-tree decomposition of the plane (see e.g. [Alberti 2008, Burr 2008, Plantinga 2004, Lorensen 1987] and references therein), whose major drawback is that the subdivision has to reach some separation bounds that are not practical so far in order to certify the results when the curve contains singular points. 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.

The other class of algorithms consists in Cylindrical-Algebraic-Decomposition (CAD) like approaches which, in short, project some characteristic points of the curve (the singular and extreme points with respect to some direction) and then lift these points back to the curve (see e.g. [Hong 1996, Gonzalez-Vega 2002, Eigenwillig 2007, Kerber 2011]). The projection (on the x-axis) is done by isolating the roots of a univariate polynomial in x: the resultant of the polynomial P defining the curve and its derivative with respect to y. Then, the lifting phase requires isolating the roots of univariate polynomials in y with *real algebraic numbers* as coefficients (the polynomial P in which x is replaced by the roots of the resultant). Roughly speaking, this can be viewed as solving a triangular system [Aubry 1999b].

In [Cheng 2010], an alternative algorithm that relies on a bivariate solver and on the computation of some multiplicities is introduced. This algorithm first computes isolating boxes around the critical points of the curves via the computation of a Rational Univariate Representation, then, a sweep line algorithm is used to connect these points with other points of the curve in order to construct an isotopic graph. This sweep line algorithm requires some multiplicities (in order to determine the topology of the curve inside the boxes of the critical points) which are given by the Rational Univariate Representation.

Chapter 2
Preliminaries

Contents

We give in this chapter the basic material that will be used in the subsequent chapters. Some of the results stated below are well known and their proofs (which we omit here) can be found in any introductory textbook on computer algebra (see for example [von zur Gathen 2003, Basu 2006, Yap 2000]). For some other results, although well-known and widely used, we were surprisingly not able to find clear references for them in the literature. This is the case, in particular for the complexity of computing greatest common divisors of univariate polynomials and also for the complexity of evaluating a polynomial at rational. For those, we provide complete and precise statements along with proofs.

The outline of this chapter is as follows. We first give in Section 2.2 the definition of the object we want to compute, that is, the Rational Univariate Representation of a bivariate system. In Section 2.3 we review the state-of-the-art complexity bounds of some fundamental operations on univariate polynomials; namely, basic operations (addition, multiplication and

Euclidean division), computing greatest common divisors and roots isolation. In section 2.4, we introduce the concept of polynomial subresultants and give some of their properties used in our algorithms, together with some complexity results. We discuss in section 2.5 modular techniques and their uses in the context of symbolic computations. We first recall the theoretical concepts behind these techniques (Modular homomorphism, Chinese Remainder Theorem) and then give some related complexity results required for the analysis of the algorithms based on modular techniques. We finally discuss an application of these modular techniques to the computation of a gcd of two univariate polynomials with integer coefficients.

2.1 Notation

We introduce below some notation that are used in the rest of this dissertation.

Bitsize. The bitsize of an integer p is the number of bits needed to represent it, that is $\lfloor \log p \rfloor + 1$ (log refers to the logarithm in base 2). If p is a rational number, then its bitsize is given as the maximum bitsize of its numerator and denominator. The bitsize of a polynomial with integer or rational coefficients is the *maximum* bitsize of its coefficients. We refer to τ_γ as the bitsize of a polynomial, rational or integer γ.

Integral domain. We denote by $\mathbb{D}$ a unique factorization domain, typically $\mathbb{Z}[X, Y]$, $\mathbb{Z}[X]$ or $\mathbb{Z}$, and by $\mathbb{F}_\mathbb{D}$ its fraction field. We also denote by $\mathbb{F}$ an arbitrary field, typically $\mathbb{Q}$ or $\mathbb{C}$.

Polynomial. For any polynomial $P \in \mathbb{D}[X]$, $Lc_X(P)$ denotes its leading coefficient with respect to the variable X (or simply $Lc(P)$ in the univariate case) and $d_X(P)$ its degree with respect to X. We denote by $\overline{P}$ the square-free part of P that is the divisor of P of maximum degree that has no square factors. An ideal generated by a set of polynomials S is denoted $\langle S \rangle$. For an ideal I, we denote by $V(I)$ its associated variety, that is, the set $\{\sigma \in \mathbb{C}^2, v(\sigma) = 0, \forall v \in I\}$. Similarly, we use $V(P)$ to refer to the set of complex roots of P. Finally, we use the notation $P \equiv 0$ when P vanishes identically.

Complexity. We use the classical O, Ω and Θ notation for denoting asymptotic upper, lower and tight bounds. For the analysis of the complexity of algorithms, we consider two types of complexities. On one hand, we consider the *arithmetic* complexity of an algorithm (in the RAM model), that is the number of arithmetic operations in some domain $\mathbb{D}$, typically $\mathbb{Z}$ or $\mathbb{Z}/\mu\mathbb{Z}$, that are performed by the algorithm. When $\mathbb{D} = \mathbb{Z}$, in order to obtain a more relevant measure of complexity, we take into account the growth of coefficients in the operations cost by considering the *bit* complexity of the algorithm, that is the number of bit operations performed by the algorithm. To easily distinguish between the arithmetic and bit complexities we use, respectively, the notation O and O_B for asymptotic upper bounds. Finally, we denote by $\widetilde{O}$ the complexities where polylogarithmic factors are omitted. Note that, unless specified otherwise, the stated complexities are worst-case complexities.

2.2 Rational Univariate Representation

A key object that we will use throughout this thesis is the Rational Univariate Representation of a bivariate system. Therefore we recall hereafter its definition and its main properties. In the following, for any polynomial $v \in \mathbb{Q}[X,Y]$ and $\sigma = (\alpha,\beta) \in \mathbb{C}^2$, we denote by $v(\sigma)$ the image of σ by the polynomial function v (e.g. $X(\alpha,\beta) = \alpha$).

Definition 2.2.1 ([Rouillier 1999]). *Let $I \subset \mathbb{Q}[X,Y]$ be a zero-dimensional ideal, $V(I) = \{\sigma \in \mathbb{C}^2, v(\sigma) = 0, \forall v \in I\}$ its associated variety, and a linear form $T = X + aY$ with $a \in \mathbb{Q}$. The RUR-candidate of I associated to $X + aY$ (or simply, to a), denoted $RUR_{I,a}$, is the following set of four univariate polynomials in $\mathbb{Q}[T]$*

$$\begin{aligned} f_{I,a}(T) &= \prod_{\sigma \in V(I)} (T - X(\sigma) - aY(\sigma))^{\mu_I(\sigma)} \\ f_{I,a,v}(T) &= \sum_{\sigma \in V(I)} \mu_I(\sigma) v(\sigma) \prod_{\varsigma \in V(I), \varsigma \neq \sigma} (T - X(\varsigma) - aY(\varsigma)), \quad \textit{for } v \in \{1, X, Y\} \end{aligned} \tag{2.1}$$

where, for $\sigma \in V(I)$, $\mu_I(\sigma)$ denotes the multiplicity of σ in I. If $(X,Y) \mapsto X + aY$ is injective on $V(I)$, we say that the linear form $X + aY$ separates $V(I)$ (or is separating for I) and $RUR_{I,a}$ is called a RUR (the RUR of I associated to a) and it defines a bijection between $V(I)$ and $V(f_{I,a}) = \{\gamma \in \mathbb{C}, f_{I,a}(\gamma) = 0\}$:

$$\begin{array}{ccc} V(I) & \to & V(f_{I,a}) \\ (\alpha,\beta) & \mapsto & \alpha + a\beta \\ \left(\dfrac{f_{I,a,X}}{f_{I,a,1}}(\gamma), \dfrac{f_{I,a,Y}}{f_{I,a,1}}(\gamma) \right) & \leftarrow\!\shortmid & \gamma \end{array}$$

Moreover, this bijection preserves the real roots and the multiplicities.

We refer to Figure 1.2 for an illustration of the one-to-one mapping defined by a RUR.

2.3 Computation with polynomials

We recall some elementary results about univariate polynomials together with the complexity of some recurrent operations. We discuss in the following three subsections, basic operations (addition, multiplication, etc), greatest common divisor, and root isolation and interval computations. We emphasize in particular on the bit complexity bounds (i.e. the number of bit operations required to perform the operations). Such a complexity depends in general on two parameters: the total degree of the polynomials and the maximum bitsize of their coefficients. It is worth noting that we only consider in the rest of this section, fast operations, that is operations with the best known complexities.

2.3.1 Basic operations

We start with the two following theorems which give both the arithmetic and the bit complexity of adding, multiplying and dividing two univariate polynomials $f, g \in \mathbb{Z}[X]$. For a detailed description of the corresponding algorithms, together with a proof of complexity, the reader may refer to [von zur Gathen 2003, Corollary 8.27, Theorem 9.6 and subsequent discussion].

Theorem 2.3.1 (addition and multiplication). *Let $f, g \in \mathbb{Z}[X]$ with degree at most d and maximum bitsize τ.*

- *The sum $f+g$ can be computed in $O(d)$ arithmetic operations, $O_B(d\tau)$ bit operations and has coefficients of bitsize at most $\tau+1$.*
- *The product $f\,g$ can be computed in $\widetilde{O}(d)$ arithmetic operations, $\widetilde{O}_B(d\tau)$ bit operations and has coefficients of bitsize in $O(\tau+\log d)$.*

Theorem 2.3.2 (Euclidean division). *Let $f, g \in \mathbb{Z}[X]$ with degree at most d and maximum bitsize τ. The computation of $q, r \in \mathbb{Z}[X]$ such that $f = qg+r$ with $\deg(r) < \deg(g)$ can be done using $\widetilde{O}(d)$ arithmetic operations and $\widetilde{O}_B(d^2\tau)$ bit operations. The polynomials q and r have degree at most d and coefficients of bitsize in $O(d\tau)$.*

Theorem 2.3.3 (Divisibility test). [1] *Let $f, g \in \mathbb{Z}[X]$ of degree at most d and maximum bitsize τ. Testing that f divides g can be done using $\widetilde{O}_B(d^2+d\tau)$ bit operations.*

---

[1] In [von zur Gathen 2003, §9.1], the authors discuss an algorithm for exact division, that is, where the remainder is known to be zero in advance, that runs in $\widetilde{O}_B(d^2+d\tau)$. Here, we assume that the division is exact, run the previous algorithm, and then, check that the quotient is correct by performing one univariate multiplication in $O(\widetilde{O}_B(d^2+d\tau))$ bit operations (Theorem 2.3.1).

We now state a bound on the complexity of evaluating a univariate polynomial which ought to be known, even though we were not able to find a proper reference for it. For completeness, we provide a proof.

Theorem 2.3.4 (evaluation). *Let a be a rational of bitsize τ_a, the evaluation at a of a univariate polynomial f of degree d and with rational coefficients of bitsize at most τ can be done in $\widetilde{O}_B(d(\tau+\tau_a))$ bit operations, while the value $f(a)$ has bitsize in $O(\tau+d\tau_a)$.*

Proof. The complexity $\widetilde{O}_B(d(\tau+\tau_a))$ is obtained by recursively evaluating the polynomial $\sum_{i=0}^{d} a_i\, x^i$ as $\sum_{i=0}^{d/2} a_i\, x^i + x^{d/2}\sum_{i=1}^{d/2} a_{i+d/2}\, x^i$. Evaluating $x^{d/2}$ at a can be done in $O_B(d\tau_a \log^3 d\tau_a)$ time by recursively computing $\log\frac{d}{2}$ multiplications of rational numbers of bitsize at most $d\tau_a$, each of which can be done in $O_B(d\tau_a \log d\tau_a \log\log d\tau_a)$ time by Schönhage-Strassen algorithm (see e.g. [von zur Gathen 2003, Theorem 8.24]. $\sum_{i=0}^{d/2} a_{i+d/2}\, a^i$ has bitsize at most $d\tau_a+\tau$, hence its multiplication with $a^{d/2}$ can be done in $O_B((d\tau_a+\tau)\log^2(d\tau_a+\tau))$ time. Hence, the total complexity of evaluating f is at most $T(d,\tau,\tau_a) = 2T(d/2,\tau,\tau_a) + O_B((d\tau_a+\tau)\log^3(d\tau_a+\tau))$. Expanding the recursive formula, we get $T(d,\tau,\tau_a) = 2^{i+1}T(\frac{d}{2^{i+1}},\tau,\tau_a) + O_B((d\tau_a + \tau)\log^3(d\tau_a+\tau)+\cdots+2^i(\frac{d}{2^i}\tau_a+\tau)\log^3(\frac{d}{2^i}\tau_a+\tau))$ which implies that $T(d,\tau,\tau_a) \leqslant O_B(d\tau_a\log^3(d\tau_a+\tau)\log d+\tau\log^3(d\tau_a+\tau)\sum_{i=0}^{\log d} 2^i)$ and hence that $T(d,\tau,\tau_a) \leqslant O_B(d(\tau_a+\tau)\log^4(d\tau_a+\tau))$ which is in $\widetilde{O}_B(d(\tau_a+\tau))$. □

2.3.2 Greatest common divisor

The greatest common divisor of two polynomials (gcd) is one of the most fundamental notions in computer algebra and appears as a sub-problem of most of the algorithms we present in this thesis. We recall the definition of the gcd of two univariate polynomials and give some of its important properties that are used throughout this work. We also give complexity results on the size and the computation of the gcd of two univariate polynomials with integer coefficients.

Definition 2.3.5 (gcd). *Let $P,Q\in\mathbb{D}[X]$. A greatest common divisor of P and Q denoted* $\gcd(P,Q)$ *is a polynomial G in $\mathbb{D}[X]$ that divides both P and Q and such that every common divisor of P and Q also divides G.*

Remark. Note that a gcd of two polynomials P and Q is defined up to an invertible element in $\mathbb{D}$. Also, one can choose a representative system in order to make the gcd unique. For example, one can require this gcd to be monic (that is with a leading coefficient equal to 1) as done for instance in [von zur Gathen 2003, §3]. In our applications, however, we are usually interested in the roots of polynomials which obviously do not change when polynomials are multiplyied by a constant factor.

When $\mathbb{D}[X]$ is an Euclidean domain, a greatest common divisor of P and Q can be computed by applying successive Euclidean divisions.

Algorithm 1 Classical Euclidean Algorithm

```
Input: $P, Q$ in $\mathbb{D}[X]$ an Euclidean domain
Output: A greatest common divisor $G \in \mathbb{D}[X]$ of $P$ and $Q$
$r_0 \leftarrow P$, $r_1 \leftarrow Q$
while $r_i \neq 0$ do
    $r_{i+1} \leftarrow r_{i-1}$ rem $r_i$
    $i \leftarrow i + 1$
return $r_{i-1}$
```

The sequence of remainders $\{r_0, r_1, \dots, r_k\}$ in the Euclidean Algorithm, is known as the Euclidean remainder sequence of P and Q and the last element of this sequence r_k is a gcd of P and Q in $\mathbb{D}[X]$.

The following result is the analog of Bézout's identity for polynomials. A proof can be found for instance in [Basu 2006, Proposition 1.9].

Theorem 2.3.6 (Bézout's identity). *Let $P, Q \in \mathbb{D}[X]$ of degrees respectively p and q. If $G = \gcd(P, Q)$ has degree g, then there exist $U, V \in \mathbb{D}[X]$ of degrees smaller respectively than $q - g$ and $p - g$ such that $UP + VQ = G$.*

The polynomials U and V are called the Bézout's coefficients of P and Q and are useful for various tasks involving the polynomials P and Q, as for example, the computation of the multiplicative inverse of P modulo Q. Indeed, suppose P and Q are relatively prime, then there exist U and V such that $UP + VQ = 1$, and hence, U is the inverse of P modulo Q. One method to compute these polynomials along with the gcd of P and Q consists in applying a variant of the Euclidean algorithm called the Extended Euclidean Algorithm. A detailed description of this basic algorithm can be found in any computer algebra textbook (see for example [von zur Gathen 2003] or [Basu 2006]).

The following result is used in the next section to exhibit the relationship between the resultant of two polynomials and their gcd (Theorem 2.4.3).

Theorem 2.3.7. *Let $P, Q \in \mathbb{D}[X]$ of degrees respectively p and q. $G = \gcd(P, Q)$ has degree larger or equal than 1 (i.e. $G \notin \mathbb{D}$) if and only if there exist $U, V \in \mathbb{D}[X]$ of degrees strictly less than q and p respectively, such that $UP + VQ = 0$.*

In addition to the gcd, we consider in the sequel the gcd-free part of P with respect to Q, that is, a divisor D of P such that $P = \gcd(P, Q)D$. Note that when $Q = \frac{\partial P}{\partial X}$, the polynomial D is the square-free part $\overline{P}$ of P. In addition, similarly as for gcds, gcd-free parts and square-free parts are defined up to an invertible factor in $\mathbb{D}$, but for convenience, we nonetheless often refer to *the* gcd-free or square-free part.

Theorem 2.3.8 ([Basu 2006, Remark 10.19]). [2] *Let P and Q in $\mathbb{D}[X]$ of degree at most d. The gcd of P and Q, or the gcd-free part of P with respect to Q can be computed with $\widetilde{O}(d)$ operations in $\mathbb{D}$.*

We now consider the case of univariate polynomials with integer coefficients. We start with the following result which is an application of the general Mignotte's bound to polynomials with integer coefficients.

Theorem 2.3.9 (**Mignotte's bound**). *[Basu 2006, Corollary 10.12] Let $P \in \mathbb{Z}[X]$ of degree d and bitsize τ and let $Q \in \mathbb{Z}[X]$ be a polynomial that divides P in $\mathbb{Z}[X]$. Then Q has bitsize in $O(d + \tau)$.*

Theorem 2.3.10. *Two polynomials P, Q in $\mathbb{Z}[X]$ with maximum degree d and bitsize at most τ have a gcd in $\mathbb{Z}[X]$ with coefficients of bitsize in $O(d+\tau)$ which can be computed with $\widetilde{O}_B(d^2\tau)$ bit operations. The same bounds hold for the bitsize and the computation of the gcd-free part of P with respect to Q.*

Proof. [Basu 2006, Corollary 10.12] states that P and Q have a gcd in $\mathbb{Z}[X]$ with bitsize in $O(d+\tau)$ while [Basu 2006, Remark 10.19] claims that a gcd and gcd-free parts of P and Q can be computed in $\widetilde{O}_B(d^2\tau)$ bit operations. These two results nevertheless, are not sufficient to conclude that a gcd of bitsize in $O(d + \tau)$ can be computed in the previous bit complexity. On the other hand, according to [Lickteig 2001, Corollary 5.2], the last non-zero Sylvester-Habicht polynomial, which is a gcd of P and Q [Basu 2006, Corollary 8.32], can be computed in $\widetilde{O}_B(d^2\tau)$ bit operations. Moreover, the same corollary proves that the Sylvester-Habicht transition matrices can be computed within the same bit complexity, which gives the cofactors of P and Q in the sequence

---

[2] Instead of the Classical Euclidean Algorithm whose complexity is quadratic in the degree of the input polynomials, Basu et al consider here the fast divide-and-conquer algorithm (also called half-gcd algorithm) introduced in [Knuth 1971] and [Schönhage 1982]. Roughly, the idea behind this algorithm is to compute only the sequence of quotients $\{q_0, q_1, \ldots, q_k\}$ (instead of the remainder sequence $\{r_0, r_1, \ldots, r_k\}$) and then, to deduce any remainder r_i (including the gcd) from this sequence. The fact that this quotient sequence is asymptotically $O(d)$ times smaller than the remainder sequence allows to decrease the asymptotic complexity of computing one element in the remainder sequence from $\widetilde{O}(d^2)$ to $\widetilde{O}(d)$. A comprehensive description of this algorithm together with a proof of complexity can be o be found in [Yap 2000, §2.4].

of the Sylvester-Habicht polynomials (i.e., $U_i, V_i \in \mathbb{Z}[X]$ such that $U_iP + V_iQ$ is equal to the i-th Sylvester-Habicht polynomials). The gcd-free part of P (resp. Q) with respect to Q (resp. P) are the cofactors corresponding to the one-after-last non-zero Sylvester-Habicht polynomial [Basu 2006, Proposition 10.14], and can thus be computed in $\widetilde{O}_B(d^2\tau)$ bit operations. However, the gcd (resp. gcd-free part) of P and Q computed this way has coefficients in $\mathbb{Z}$ of bitsize in $O(d\tau)$. To obtain a gcd with the stated bitsize i.e. $O(d+\tau)$, one can divide the last non-zero Sylvester-Habicht polynomial by the gcd of its coefficients. This yields the gcd (resp. the gcd-free part) of P and Q of smallest bitsize in $\mathbb{Z}[X]$ which is known to be in $O(d+\tau)$ according to Theorem 2.3.9. The gcd of the coefficients, which are of bitsize $\widetilde{O}(d\tau)$ [Basu 2006, Proposition 8.46], follows from $O(d)$ gcds of two integers of bitsize $\widetilde{O}(d\tau)$ and each such gcd can be computed with $\widetilde{O}_B(d\tau)$ bit operations [Yap 2000, §2.A.6]. Therefore, a gcd (resp. gcd-free part) of P and Q of bitsize $O(d+\tau)$ can be computed in $\widetilde{O}_B(d^2\tau)$ bit complexity. □

We provide a more refined version of the previous result in the case of two polynomials with different degrees and bitsizes.

Theorem 2.3.11. *Let P and Q be two polynomials in $\mathbb{Z}[X]$ of degrees p and q and of bitsizes τ_P and τ_Q, respectively. A gcd of P and Q of bitsize $O(\min(p+\tau_P, q+\tau_Q))$ in $\mathbb{Z}[X]$, can be computed in $\widetilde{O}_B(\max(p,q)(p\tau_Q+q\tau_P))$ bit operations. A gcd-free part of P with respect to Q, of bitsize $O(p+\tau_P)$ in $\mathbb{Z}[X]$, can be computed in the same bit complexity.*

Proof. The algorithm in [Lickteig 2001] uses the well-known half-gcd approach to compute any polynomial in the Sylvester-Habicht and cofactors sequence in a softly-linear number of arithmetic operations, and it exploits Hadamard's bound on determinants to bound the size of intermediate coefficients. When the two input polynomials have different degrees and bitsizes, Hadamard's bound reads as $\widetilde{O}(p\tau_Q + q\tau_P)$ instead of simply $\widetilde{O}(d\tau)$ and, similarly as for Theorem 2.3.10, the algorithm in [Lickteig 2001] yields a gcd and gcd-free parts of P and Q in $\widetilde{O}_B(\max(p,q)(p\tau_Q + q\tau_P))$ bit operations. Furthermore, the gcd and gcd-free parts computed this way are in $\mathbb{Z}[X]$ with coefficients of bitsize $\widetilde{O}(p\tau_Q + q\tau_P)$, thus, dividing them by the gcd of their coefficients can be done with $\widetilde{O}_B(\max(p,q)(p\tau_Q + q\tau_P))$ bit operations and yields a gcd and gcd-free parts in $\mathbb{Z}[X]$ with minimal bitsize, which is as claimed by Mignotte's bound (Theorem 2.3.9) in $O(p+\tau_P)$. □

2.3.3 Root isolation and interval arithmetic

We recall some bounds on univariate polynomial roots and their separation (for a single root and also amortized over all the roots), the complexity of isolating the real roots of a univariate polynomial, and elementary results on interval arithmetic. In the following, f denotes a univariate polynomial of degree d with integer coefficients of bitsize at most τ.

Theorem 2.3.12 ([Yap 2000, §6.2 Lemma 6.5)] and [Basu 2006, Proposition 10.9]). *For any root γ of f, $\max(1,|\gamma|)=2^{O(\tau)}$ and $\prod_{\{\gamma \text{ root of } f\}}\max(1,|\gamma|)=2^{O(\tau)}$.*

Theorem 2.3.13 (separation bound)**.** *([Emiris 2010, Theorem 1]) Let sep_γ be the separation bound of a root γ of f defined as the minimum distance between γ and any other (possibly complex) root of f. Then $sep_\gamma=2^{-\widetilde{O}(d\tau+d^2)}$ and $\prod_{\{\gamma \text{ roots of } f\}} sep_\gamma=2^{-\widetilde{O}(d\tau+d^2)}$.*

Theorem 2.3.14. *Let f and g be integer polynomials of degree bounded by d and bitsize bounded by τ. Assume f square free and coprime with g, then for any root γ of f, $|g(\gamma)|=2^{-O(d\tau)}$ and $\prod_{\{\gamma \text{ root of } f\}}|g(\gamma)|=2^{-O(d\tau)}$.*

Proof. The bound for a single root is stated in [Yap 2000, Lemma 6.34]. For the bound on all the roots, one remarks that the resultant $res(f,g)$ of f and g can be written (see for instance [Yap 2000, Theorem 6.15]) as $res(f,g)=\mathrm{Lc}(f)^d\prod_{\{\gamma \text{ root of } f\}}|g(\gamma)|$. Since f and g are coprime this resultant is a non-nul integer. Also $\mathrm{Lc}(f)$ is bounded by 2^τ and it follows that $\prod_{\{\gamma \text{ root of } f\}}|g(\gamma)|=|\frac{res(f,g)}{\mathrm{Lc}(f)^d}|\geqslant|\frac{1}{\mathrm{Lc}(f)^d}|\geqslant 2^{-d\tau}$. ☐

Theorem 2.3.15 (real roots isolation)**.** *([Mehlhorn 2013, Theorem 5][3]) Isolating intervals of all the real roots of f can be computed and refined up to a width less than 2^{-L} with $\widetilde{O}_B(d^3+d^2\tau+dL)$ bit operations.*

Moreover, we assume that all isolating intervals of a root γ of f are dyadic intervals, that is of the form $J_\gamma=[\lfloor\gamma\rfloor+m/2^k,\lfloor\gamma\rfloor+(m+1)/2^k]$ with $k>0$ and m an integer between 0 and 2^k-1. It follows that the width $w(J_\gamma)=2^k$ and the maximum bitsize of the endpoints of the interval J_γ is in $O(\tau+k)$ (indeed, $|\gamma|\leqslant 2||f||_\infty\leqslant 2^{\tau+1}$ (see for instance [Yap 2000, §6.2 Lemma 6.5)]) and hence $\tau(\lfloor\gamma\rfloor)\leqslant\tau+2$ and $\tau(J_\gamma)=\tau((\lfloor\gamma\rfloor 2^k+m+1)/2^k)\leqslant\tau+k+4)$.

For evaluating a polynomial on an interval we use interval arithmetic and the fast divide and conquer scheme as explained in the proof of Theorem 2.3.4. The interval resulting of the evaluation of the polynomial f on an interval J using these rules is denoted $\Box f(J)$. The next lemma bounds the growth of

---

[3] Theorem 5 of [Mehlhorn 2013] is stated for complex roots, however it is straightforward to identify the boxes containing the real roots within the same complexity. Indeed, by considering L in $\widetilde{O}(d\tau+d^2)$ with 2^{-L} smaller than twice the root separation bound of f (which is possible by Theorem 2.3.13), the isolating boxes of the complex roots do not intersect the real axis.

the width of an interval when a polynomial evaluation is performed. Note that the assumption that the width is smaller that 1 is not restrictive for an isolating interval of a root γ of f since this is always the case for our choice of dyadic representation.

Theorem 2.3.16. *Let J be an interval of width smaller than 1. Then, for each $\gamma \in J$ and each $y \in \Box f(J)$, we have $|y - f(\gamma)| \leqslant 2^{2d+\tau} \max(1, |\gamma|)^{d-1} w(J)$.*

Proof. The fast evaluation scheme for the polynomial $f(x) = \sum_{i=0}^{d} a_i\, x^i$ is recursively computed as $\sum_{i=0}^{d/2} a_i\, x^i + x^{d/2} \sum_{i=1}^{d/2} a_{i+d/2}\, x^i$. Let denote $f_1(x) = \sum_{i=0}^{d/2} a_i\, x^i$, $f_2(x) = \sum_{i=1}^{d/2} a_{i+d/2}\, x^i$, $\varepsilon = w(J)$ and $m = \max(1, |\gamma|)$. The proof proceeds by induction on the degree $d = 2^k$ for the formula $|y - f(\gamma)| \leqslant 2^{2d+\tau} m^{d-1} \varepsilon$.

For $d = 2^0 = 1$, $y = a_0 + a_1(\gamma + e_1)$ with $|e_1| \leqslant \varepsilon$ and $f(\gamma) = a_0 + a_1\gamma$, hence $|y - f(\gamma)| = |a_1 e_1| \leqslant 2^\tau \varepsilon$ which proves the formula.

Now assume $d = 2^k$ for $k \geqslant 1$, one can write $y \in \Box f(J) = \Box f_1(J) + J^{d/2} \Box f_2(J)$ as $y = y_1 + (\gamma + e_1) \dots (\gamma + e_{d/2}) y_2$ with $y_i \in \Box f_i(J)$, $|e_j| \leqslant \varepsilon$ and $f(\gamma) = f_1(\gamma) + \gamma^{d/2} f_2(\gamma)$. One has

$$|y - f(\gamma)| \leqslant |y_1 - f_1(\gamma)| + |\gamma|^{d/2} |y_2 - f_2(\gamma)| + |y_2| |(\gamma + e_1) \dots (\gamma + e_{d/2}) - \gamma^{d/2}| \tag{2.2}$$

Using the induction, $|y_i - f_i(\gamma)| \leqslant 2^{d+\tau} m^{d/2-1} \varepsilon$ and $|\gamma|^{d/2}| \leqslant m^{d/2}$. The triangular inequality yields $|y_2| \leqslant |y_2 - f_2(\gamma)| + |f_2(\gamma)|$, hence using the induction and the straightforward upper bound $|f_2(\gamma)| \leqslant 2^\tau m^{d/2}(d/2 + 1)$, we obtain $|y_2| \leqslant 2^\tau m^{d/2}(2^{d/2}\varepsilon/m + d/2 + 1) \leqslant 2^\tau m^{d/2}(2^{d/2} + d/2 + 1)$.

On the other hand,

$$\begin{aligned}
|(\gamma + e_1) \dots (\gamma + e_{d/2}) - \gamma^{d/2}| &= |\sum_{j=1}^{j=d/2-1} e_j \gamma^{j-1} (\gamma + e_{j+1}) \dots (\gamma + e_{d/2})| \\
&\leqslant \sum_{j=1}^{j=d/2-1} \varepsilon |\gamma|^{j-1} |\gamma + e_{j+1}| \dots |\gamma + e_{d/2}| \\
&\leqslant (d/2 - 1) \varepsilon m^{d/2-1} 2^{d/2-1}
\end{aligned}$$

since $|\gamma + e_k| \leqslant 2m$.

With all these results together, Eq. (2.2) yields $|y - f(\gamma)| \leqslant 2^{2d+\tau} \varepsilon m^{d-1} (2^{1-d} + 2^{-3/2d-1}(d/2 - 1)(2^d + d/2 + 1))$ and one can show that the factor $2^{1-d} + 2^{-3/2d-1}(d/2 - 1)(2^d + d/2 + 1)$ is smaller than 1 for $d \geqslant 1$. ☐

2.4 Subresultant sequence

We review in this section the theory of subresultants which is an ubiquitous tool in computer algebra for solving algebraic systems. We first introduce the resultant of two polynomials which is a polynomial closely related to the gcd and then discuss the subresultant sequence which generalizes the notion of resultant as the remainder sequence generalizes the notion of gcd.

2.4.1 Resultant

Let P and Q be two non-zero polynomials of degree p and q in $\mathbb{D}[X]$:

$$P = a_p X^p + a_{p-1} X^{p-1} + \ldots + a_0 \quad \text{and} \qquad Q = b_q X^q + b_{q-1} X^{q-1} + \ldots + q_0.$$

We define the Sylvester matrix of P and Q and describe the link between this matrix and Euclid's algorithm.

Definition 2.4.1 (Sylvester matrix). *The Sylvester matrix of P and Q is the following $(p+q)$-square matrix whose rows are $X^{q-1}P, \ldots, P, X^{p-1}Q, \ldots, Q$ considered as vectors in the basis $X^{p+q-1}, \ldots, X, 1$.*

$$\mathrm{Syl}(P,Q) = \overbrace{\begin{pmatrix}
a_p & a_{p-1} & \cdots & \cdots & a_0 & & & \\
& a_p & a_{p-1} & \cdots & \cdots & a_0 & & \\
& & \ddots & & & & \ddots & \\
& & & a_p & a_{p-1} & \cdots & \cdots & a_0 \\
b_q & b_{q-1} & \cdots & b_0 & & & & \\
& b_q & b_{q-1} & \cdots & b_0 & & & \\
& & \ddots & & & \ddots & & \\
& & & \ddots & & & \ddots & \\
& & & & b_q & b_{q-1} & \cdots & b_0
\end{pmatrix}}^{p+q \ \textit{columns}}
\begin{matrix}
\left.\vphantom{\begin{matrix}a\\a\\a\\a\end{matrix}}\right\} q \ \textit{rows} \\
\left.\vphantom{\begin{matrix}b\\b\\b\\b\\b\end{matrix}}\right\} p \ \textit{rows}
\end{matrix}$$

Definition 2.4.2 (Resultant). *The resultant of the polynomials P and Q with respect to X, denoted $Res_X(P,Q)$ or $Res(P,Q)$ is the determinant of the Sylvester matrix* $\mathrm{Syl}(P,Q)$*.*

The matrix $\mathrm{Syl}(P,Q)$ is the transpose of the matrix of the linear mapping

$$(U,V) \longmapsto U\,P + V\,Q$$

where $U = u_{q-1}X^{q-1} + \ldots + u_0$, $V = v_{p-1}X^{p-1} + \ldots + v_0$ and the couple (U,V) is given as $(u_{q-1}, \ldots, u_0, v_{p-1}, \ldots, v_0)$ and where $UP + VQ$ is identified to its vector of coefficients.

It is known from Theorem 2.3.7, that P and Q have a non-trivial common factor in $\mathbb{D}[X]$ if and only if there exist two polynomials U and V of degree strictly less than q and p respectively, such that $U\,P + V\,Q = 0$. Translated in terms of Sylvester matrix, this result yields the following property of the resultant.

Theorem 2.4.3. *$Res(P,Q) = 0$ if and only if P and Q have a common factor in $\mathbb{D}[X]$.*

Proof. The statement directly follows from the definition of the resultant as the determinant of Syl(P,Q) and from the above discussion. □

The definition of the resultant as the determinant of the Sylvester matrix, yields an important property called **specialization**. Let $\phi : \mathbb{D} \longrightarrow \mathbb{D}'$ be a ring homomorphism. Note that ϕ induces a homomorphism from $\mathbb{D}[X] \longrightarrow \mathbb{D}'[X]$ by mapping all the coefficients of the polynomial. We will refer to the latter also as ϕ.

Theorem 2.4.4. *Let $a_p, b_q \in \mathbb{D}$ be the leading coefficients of P and Q respectively. If $\phi(a_p) \not\equiv 0$ and $\phi(b_q) \not\equiv 0$, then $\phi(Res(P,Q)) = Res(\phi(P), \phi(Q))$.*

Let now consider the case where P and Q are bivariate polynomials with integer coefficients. Considering these polynomials as elements in $(\mathbb{Z}[X])[Y]$, the Sylvester matrix has now as entries, polynomials in $\mathbb{Z}[X]$ and its determinant which is the resultant of P and Q is also in $\mathbb{Z}[X]$. The following proposition is important for the algorithms presented in this thesis. See [Cox 1997, §6] for a more general statement.

Proposition 2.4.5. *Let $P, Q \in \mathbb{Z}[X,Y]$ be coprime and $Res_Y(P,Q)$ their resultant with respect to Y. Let $a_p(X)$ and $b_q(X)$ be their leading coefficients in Y. The two following statements are equivalent.*

- *$\alpha \in \overline{\mathbb{Q}}$ is a root of $Res_Y(P,Q)$.*
- *$a_p(\alpha) = b_q(\alpha) = 0$, or there exits $\beta \in \overline{\mathbb{Q}}$ such that $P(\alpha,\beta) = Q(\alpha,\beta) = 0$.*

Given two bivariate polynomials P and $Q \in \mathbb{Z}[X,Y]$, the previous result shows that computing their resultant with respect to Y yields the X-coordinate of all their common solutions. A natural further step would be to compute for each common solution the corresponding Y-coordinate. Such information can be obtained using the concept of subresultant polynomials which we introduce in the following subsection.

2.4.2 Subresultant sequence

We first introduce the concept of *polynomial determinant* of a matrix which is used in the definition of subresultants.

Definition 2.4.6 (Polynomial determinant). *Let M be an $m \times n$ matrix with $m \leqslant n$ and M_i be the square sub-matrix of M consisting of the first $m-1$ columns and the i-th column of M, for $i = m, \dots, n$. The* polynomial determinant *of M is the polynomial defined as* $\det(M_m)X^{n-m} + \det(M_{m+1})X^{n-(m+1)} + \dots + \det(M_n)$.

Let P and Q be two non-zero polynomials in $\mathbb{D}[X]$ of degree p and q with $p > q$. As for the resultant, subresultants are closely related to the Sylvester matrix of P and Q.

Definition 2.4.7 (Sylvester sub-matrix). *For $i = 0, \dots, \min(q, p-1)$, the i-th Sylvester sub-matrix of P and Q is the $(p+q-2i) \times (p+q-i)$ matrix extracted from $\mathrm{Syl}(P,Q)$ by deleting the i last rows of the coefficients of P, the i last rows of the coefficients of Q, and the i last columns.*

$$\mathrm{Syl}_i(P,Q) = \begin{array}{c} \overbrace{\hphantom{a_p \quad a_{p-1} \quad \cdots \quad \cdots \quad a_0 \quad \cdots \quad a_0}}^{\textit{p+q - i columns}} \\ \left(\begin{array}{ccccccc} a_p & a_{p-1} & \cdots & \cdots & a_0 & & \\ & \ddots & & & & \ddots & \\ & & a_p & a_{p-1} & \cdots & \cdots & a_0 \\ b_q & b_{q-1} & \cdots & b_0 & & & \\ & \ddots & & & \ddots & & \\ & & \ddots & & & \ddots & \\ & & & b_q & b_{q-1} & \cdots & b_0 \end{array}\right) \end{array} \begin{array}{l} \\ \left.\vphantom{\begin{array}{c}a\\a\\a\end{array}}\right\} \textit{q-i rows} \\ \left.\vphantom{\begin{array}{c}a\\a\\a\\a\end{array}}\right\} \textit{p-i rows} \end{array}$$

Definition 2.4.8 (Polynomial subresultant). *For $i = 0, \dots, \min(q, p-1)$, the i-th polynomial subresultant of P and Q, denoted by $Sres_{X,i}(P,Q)$ is the polynomial determinant of $\mathrm{Syl}_i(P,Q)$. When $q = p$, the q-th polynomial subresultant of P and Q is $b_q^{-1}Q$.*[4]

Using the notation of Definition 2.4.6, every polynomial subresultant can be written as $Sres_{X,i}(P,Q) = \det(\mathrm{Syl}_{i,p+q-2i})X^i + \dots + \det(\mathrm{Syl}_{i,p+q-i})$ where $\mathrm{Syl}_{i,k}$ denotes the square sub-matrix of $\mathrm{Syl}_i(P,Q)$ consisting of the first $p+q-i-1$ columns and the k-th column of $\mathrm{Syl}_i(P,Q)$. Note that when $i = 0$, Syl_0 coincides with the Sylvester matrix of P and Q and the corresponding polynomial determinant then becomes the determinant of $\mathrm{Syl}(P,Q)$ which is the resultant of P and Q.

---

[4] It can be observed that, when $p > q$, the q-th subresultant is equal to $b_q^{p-q-1}Q$, however it is not defined when $p = q$. In this case, following El Kahoui [El Kahoui 2003], we extend the definition to $b_q^{-1}Q$ assuming that the domain $\mathbb{D}$ is integral, which is the case throughout this thesis.

Definition 2.4.9 (Principal subresultant). *$Sres_{X,i}(P,Q)$ has degree at most i in X, and the coefficient of its monomial of degree i in X, which we denote by $sres_{X,i}(P,Q)$, is called the i-th* principal subresultant coefficient*.*

It may happen that $sres_{X,i}(P,Q)$ equals zero; in that case, we say that the polynomial subresultant $Sres_{X,i}(P,Q)$ is **defective**, otherwise $Sres_{X,i}(P,Q)$ is said to be **non-defective** or **regular**.

The principal subresultant $sres_{X,i}(P,Q)$ is the determinant of the matrix $\mathrm{Syl}_{i,p+q-2i}$ which is represented by the linear mapping

$$(U,V) \longmapsto U\,P + V\,Q$$

where $U = u_{q-1-i}X^{q-1-i} + \ldots + u_0$, $V = v_{p-1-i}X^{p-1_i} + \ldots + v_0$ and the couple (U,V) is given as $(u_{q-1-i},\ldots,u_0,v_{p-1-i},\ldots,v_0)$.

The following lemma is a direct consequence of the previous observation.

Lemma 2.4.10. *$sres_{X,i}(P,Q) = 0$ if and only if there exist non-zero polynomials $U,V \in \mathbb{D}[X]$ with $\deg(U) < q-i$ and $\deg(V) < p-i$, such that $\deg(UP+VQ) < i$.*

One can examine when the principal subresultants vanish to identify the degrees that appear in the remainder sequence of the Euclidean algorithm. In particular the following theorem shows how to determine the degree of the gcd of P and Q depending on the vanishing of some principal subresulant coefficients.

Theorem 2.4.11. *[Basu 2006, Proposition 4.25] The degree of $\gcd(P,Q)$ is equal to k if and only if $sres_0 = sres_1 = \ldots = sres_{k-1} = 0$ and $sres_k \neq 0$.*

As for the resultant (Theorem 2.4.4), the matricial definition of the polynomial subresultants yields the same specialization property.

Theorem 2.4.12. *Let $a_p, b_q \in \mathbb{D}$ be the leading coefficients of P and Q respectively. If $\phi(a_p) \not\equiv 0$ and $\phi(b_q) \not\equiv 0$, then, for all i, $\phi(Sres_i(P,Q)) = Sres_i(\phi(P),\phi(Q))$.*

We now state the following fundamental theorem called the gap structure theorem for subresultants. It describes the particular structure of the polynomial subresultants graphically displayed in Figure 2.1 and shows the link between these polynomials and the polynomials appearing in the Euclidean remainder sequence. We refer to [El Kahoui 2003, Theorem 4.3] for a concise proof of this theorem.

Theorem 2.4.13 (Gap Structure Theorem). *Let $0 \leqslant j \leqslant min(q,p-1)$, and assume that $Sres_j$ is regular and $Sres_{j-1} \neq 0$ is of degree $k < j-1$. Then:*

(i) $Sres_{j-2} = \ldots = Sres_{k+1} = 0$

(ii) $sres_j^{j-k-1} Sres_k = sres_{j-1,k}^{j-k-1} Sres_{j-1}$

(iii) $sres_j^2 Sres_{k-1} = (-1)^{j-k} sres_{j-1,k} sres_k Sres_j + C_j Sres_{j-1}$ *with* $C_j \in \mathbb{D}[X]$

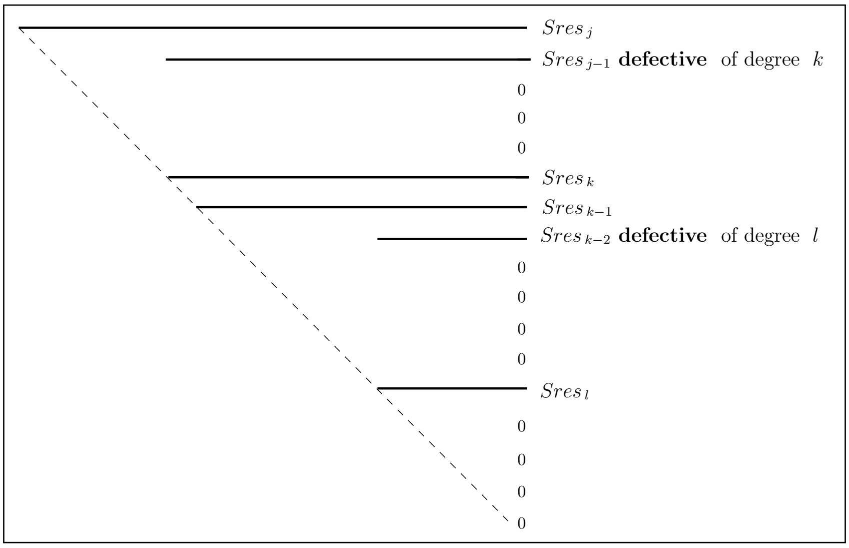
$Sres_j$
$Sres_{j-1}$ **defective** of degree k
0
0
0
$Sres_k$
$Sres_{k-1}$
$Sres_{k-2}$ **defective** of degree l
0
0
0
0
$Sres_l$
0
0
0
0

Figure 2.1: Illustration of the Gap Structure Theorem.

A direct consequence of Theorem 2.4.13 is the following corollary whose proof can be found for instance in [Basu 2006, Corollary 8.34].

Corollary 2.4.14. *If* $r_{j-1}(P, Q)$ *and* $r_j(P, Q)$ *are two successive polynomials in the Euclidean remainder sequence of* P *and* Q*, of degrees respectively* d_{j-1} *and* d_j*, then* $Sres_{d_{j-1}-1}(P, Q)$ *and* $Sres_{d_j}(P, Q)$ *are proportional to* $r_j(P, Q)$*.*

In fact, the previous corollary together with the gap structure theorem show that the subresultants of P and Q are equal either to 0 or to polynomials in the Euclidean Remainder Sequence of P and Q (up to multiplicative factors in $\mathbb{D}$). In particular, one important property which we will often use in the rest of this thesis is that the last non-zero subresultant of P and Q is non-defective and is a greatest common divisor of P and Q.

Let now reconsider the case of two bivariate polynomials with integer coefficients, $P = \sum_{i=0}^{p} a_i Y^i$ and $Q = \sum_{i=0}^{q} b_i Y^i$ in $\mathbb{D}[X, Y]$, with $p \geqslant q$. As shown by Proposition 2.4.5, one can identify the X-coordinates of the common solutions of P and Q by computing their resultant with respect to Y. Using the polynomial subresultant defined above, it is also possible to determine the gcd of P and Q at these X-coordinates. This key feature, stated in the following theorem is instrumental in the triangular decomposition algorithms described in Chapters 3 and 5. Note that this result is often stated with the stronger assumption that is, that *none* of the leading terms $a_p(\alpha)$ and $b_q(\alpha)$ vanish. This property is a direct consequence of the specialization property of subresultants and of the gap structure theorem; see [El Kahoui 2003, Lemmas 2.3, 3.1 and Corollary 5.1] for a proof.

Theorem 2.4.15. *For any α such that $a_p(\alpha)$ and $b_q(\alpha)$ do not both vanish, the first $Sres_{Y,k}(P,Q)(\alpha,Y)$ (for k increasing) that does not identically vanish is of degree k and it is the gcd of $P(\alpha,Y)$ and $Q(\alpha,Y)$ (up to a nonzero constant in the fraction field of $\mathbb{D}(\alpha)$).*

2.4.3 Subresultants computation

Exploiting the gap structure theorem (Theorem 2.4.13), one can compute efficiently the polynomial subresultants using a variant of the classical Euclidean algorithm (see e.g. [Basu 2006, Algorithm 8.21]). This algorithm performs successively divisions and returns the sequence of the intermediate remainders consisting in the sequence of the polynomial subresultants. Furthermore, the latter approach allows to adapt fast techniques for computing one polynomial in the Euclidean remainder sequence [Yap 2000, §2.4] in order to compute any polynomial subresultant without computing the complete sequence (see [Reischert 1997] and [Lickteig 2001]).

Next, we give some bit complexity results related to the size and the computation of the polynomial subresultants of two polynomials with integer coefficients.

We first analyze the size of the polynomial subresultants. Since these polynomials are defined by determinants of some Sylvester sub-matrices, we can use the Hadamard's inequality ([von zur Gathen 2003, Theorem 16.6]) to derive bounds on the degree and the size of their coefficients. Proofs of the following theorems can be found in [Basu 2006, Propositions 8.11,8.12 and 8.34].

Theorem 2.4.16. *Let P and Q be in $\mathbb{Z}[X_1,\ldots,X_n][Y]$ (n fixed) with coefficients of bitsize at most τ such that their degrees in Y are bounded by d_Y and their degrees in the other variables are bounded by d.*

- *The coefficients of $Sres_{Y,i}(P,Q)$ have bitsize in $\widetilde{O}(d_Y\tau)$.*
- *The degree in X_j of $Sres_{Y,i}(P,Q)$ is at most $2d(d_Y-i)$.*

Theorem 2.4.17. *Let P and Q be in $\mathbb{Z}[X_1, \ldots, X_n][Y]$ (n fixed) with coefficients of bitsize at most τ such that their degrees in Y are bounded by d_Y and their degrees in the other variables are bounded by d.*

- *Any polynomial subresultant $Sres_{Y,i}(P,Q)$ can be computed in $\widetilde{O}(d^n d_Y^{n+1})$ arithmetic operations, and $\widetilde{O}(d^n d_Y^{n+2}\tau)$ bit operations.*

- *All the polynomial subresultants can be computed in $\widetilde{O}(d^n d_Y^{n+2})$ arithmetic operations, and $\widetilde{O}(d^n d_Y^{n+3}\tau)$ bit operations.*

2.5 Modular techniques

The use of modular techniques is rather classical in computer algebra to avoid certain computational problems that arise in general with symbolic computations. Roughly, the principle of these techniques consists in transforming one problem over an algebraic domain, to a set of similar problems over simpler domains. A classical example, on which we will focus our attention, is the approach based on the integer *Chinese Remainder Theorem* or *CRT*, where instead of directly solving a problem over $\mathbb{Z}$, one solves it modulo several integers m (over $\mathbb{Z}_m$) and then, reconstruct the solution over $\mathbb{Z}$ from the set of solutions over $\mathbb{Z}_m$.

An important advantage of such an approach is that it allows to avoid intermediate coefficient swell during the computations. This swelling phenomenon is quite well observable for instance in the computation of a greatest common divisor of two univariate polynomials with integer coefficients, where the size of the intermediate coefficients can reach a bound in $\widetilde{O}(d^2\tau)$ [von zur Gathen 2003, Theorem 6.52] whereas the size of the coefficients in the computed gcd does not exceed a quantity in $\widetilde{O}(d+\tau)$ (Theorem 2.3.9).

Historically, the use of modular techniques in computer algebra can be traced back to Collins [Collins 1971] and Brown [Brown 1971] in the context of computing resultants of multivariate polynomials, gcds and subresultants. Since then, an extensive literature has been produced on the subject; see for example [Yap 2000, §4] or [von zur Gathen 2003, §5] for comprehensive introductions.

Next, we review in short the principle of the *CRT*-based algorithms for solving algebraic problems and recall some related complexity results. We then discuss briefly the application of this principle to the computation of the gcd of two univariate polynomials with integer coefficients.

2.5.1 General principle

We first define the concept of modular homomorphism which is of primary importance for the design of the modular algorithm presented below.

Definition 2.5.1. *The homomorphism of the reduction modulo* $m \in \mathbb{Z}$ *is the map:* $\phi_m : \mathbb{Z} \to \mathbb{Z}_m$ *defined by* $\forall x \in \mathbb{Z} : \phi_m(x) = x \mod m$.

We extend the previous definition to modular homomorphisms over the ring of polynomials with integer coefficients and keep the same notation as above. More precisely, given a polynomial $P \in \mathbb{Z}[x_1, \ldots, x_n]$, we denote by $\phi_m(P)$ the polynomial resulting from the reduction of all the integer coefficients of P by ϕ_m. For example, if $P(x, y) = 7xy^2 + 9xy + 13x + 15y + 1$ then $\phi_5(P(x, y)) = 2xy^2 + 4xy + 3x + 1$.

An important property of this modular homomorphism is that it allows, under some conditions, to reconstruct from a set of images in $\mathbb{Z}_{m_1}, \ldots, \mathbb{Z}_{m_k}$ a unique preimage in $\mathbb{Z}$. This key feature is formalized in the following theorem known as the *Chinese Remainder Theorem* or more shortly *CRT*.[5] See [Yap 2000, Theorem 4.1] or [von zur Gathen 2003, Corollary 5.3] for proofs.

Theorem 2.5.2. *Let* $m_1, m_2, \ldots, m_k \in \mathbb{Z}$ *be integers that are pairwise coprime, that is,* $\gcd(m_i, m_j) = 1$ *for* $i \neq j$ *and let* $m = \prod_{i=1}^{k} m_i$. *Then, for any sequence of integers* $r_1, r_2, \ldots, r_k$ *with* $r_i \in \mathbb{Z}_{m_i}$, *there exists a unique* $r \in \mathbb{Z}_m$ *such that:*

$$r \equiv r_i \mod m_i \text{ for } i = 1, \ldots, k$$

The proof of Theorem 2.5.2 is constructive and yields a simple algorithm for computing the unique solution r of the set of modular equations $x \equiv r_i \mod m_i$ for $i = 1, \ldots, k$ [von zur Gathen 2003, Algorithm 5.4]. Moreover, an efficient version of this algorithm based on a "divide-and-conquer" strategy with a quasi-optimal complexity is described in [von zur Gathen 2003, §10.3]. In the following, we refer to this algorithm as the *Chinese Remainder Algorithm* and give its complexity in Theorem 2.5.4.

Using the ingredients above, we are now able to outline the different steps of the *CRT*-based algorithm for solving problems defined over the polynomial ring $\mathbb{Z}[x_1, \ldots, x_n]$ (See the illustration in Figure 2.2). This algorithm consists actually of three steps. First, modular homomorphisms are applied using several moduli m_i in order to obtain a set of similar problems over $\mathbb{Z}_{m_i}[x_1, \ldots, x_n]$,

---

[5] Note that a more general version of this theorem can be found in the literature where instead of integers, one consider ideals (see e.g. [Yap 2000, Theorem 4.1]).

then, the solutions of these problems are computed over $\mathbb{Z}_{m_i}[x_1, \ldots, x_n]$ and finally, these solutions are lifted back over $\mathbb{Z}[x_1, \ldots, x_n]$ using the Chinese Remainder Algorithm.

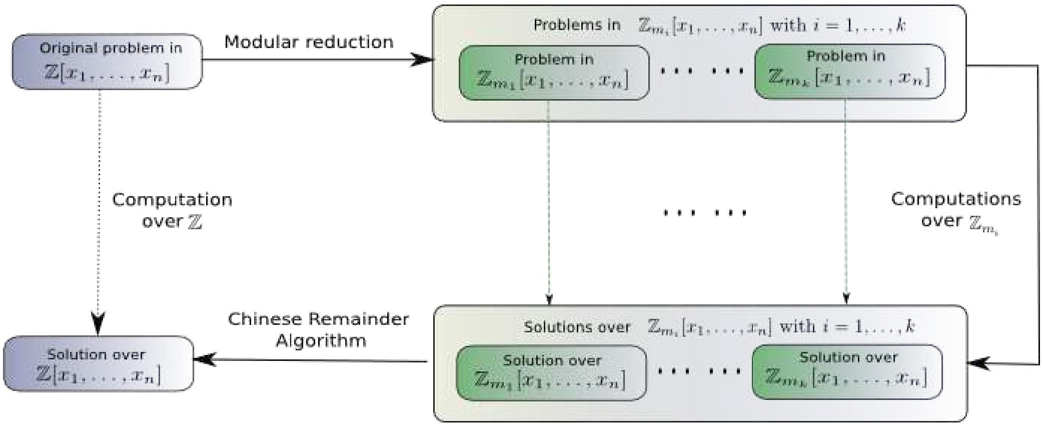
Original problem in $\mathbb{Z}[x_1, \ldots, x_n]$
Modular reduction
Problems in $\mathbb{Z}_{m_i}[x_1, \ldots, x_n]$ with $i = 1, \ldots, k$
Problem in $\mathbb{Z}_{m_1}[x_1, \ldots, x_n]$
Problem in $\mathbb{Z}_{m_k}[x_1, \ldots, x_n]$
Computation over $\mathbb{Z}$
Computations over $\mathbb{Z}_{m_i}$
Chinese Remainder Algorithm
Solutions over $\mathbb{Z}_{m_i}[x_1, \ldots, x_n]$ with $i = 1, \ldots, k$
Solution over $\mathbb{Z}_{m_1}[x_1, \ldots, x_n]$
Solution over $\mathbb{Z}_{m_k}[x_1, \ldots, x_n]$
Solution over $\mathbb{Z}[x_1, \ldots, x_n]$

Figure 2.2: General scheme of the *CRT*-based approach.

An important issue in the *CRT*-based approach concerns the correctness of the lifted solution. Indeed, in order to apply correctly the *Chinese Remainder Algorithm*, two conditions must be satisfied.

- For every chosen moduli m_i, the solution of the modular image over $\mathbb{Z}_{m_i}[x_1, \ldots, x_n]$ has to be equal to the modular image of the solution over $\mathbb{Z}[x_1, \ldots, x_n]$ by the homomorphism ϕ_{m_i}. Homomorphisms for which this equality does not hold are called *unlucky* homomorphisms and a non-trivial task in the design of a *CRT*-based algorithm is to detect and to discard them. For example, in the modular computation of the gcd of two univariate polynomials described in the next section, the *unlucky* homomorphisms are those for which the degree of this gcd increases.
- The product of the chosen moduli m_i has to be larger than the maximum integer appearing in the solution, so that all the coefficients of the solution can be recovered from their images modulo the m_i. In general, this is done by computing (when possible) a theoretical bound on the size of the solution and choosing a set of moduli whose product is larger than those bounds. In some cases however, such bounds turn out to be hard to compute or quite pessimistic which affects the efficiency of the algorithm.

In the following, we give two important complexity results that are needed for the analysis of a *CRT*-based algorithm.

Theorem 2.5.3. *[von zur Gathen 2003, Theorem 10.24] Given* $m_1, \ldots, m_k \in \mathbb{N}_{\geqslant 2}$ *with* $m = \prod_{i=1}^{k} m_i$ *of bitsize* τ *and* $r \in \mathbb{N}$ *less than* m*, we can compute* $r \mod m_1, \ldots, r \mod m_k$ *using* $\widetilde{O}_B(\tau)$ *bit operations.*

Theorem 2.5.4. *[von zur Gathen 2003, Theorem 10.25] Given pairwise coprime integers* $m_1, \ldots, m_k \in \mathbb{N}_{\geqslant 2}$ *with* $m = \prod_{i=1}^{k} m_i$ *of bitsize* τ *and* $r_1, \ldots, r_k \in \mathbb{N}$ *such that* $r_i < m_i$ *for all* i*, we can compute the unique solution* $r \in \mathbb{N}$ *less than* m *of the Chinese Remainder Problem* $r \equiv r_i \mod m_i$ *for* $i = 1, \ldots, k$ *using* $\widetilde{O}_B(\tau)$ *bit operations.*

Remark. In practice, in order to avoid the cost of checking the coprimality of the moduli m_i (hypothesis of the Chinese Remainder Theorem), the m_i are chosen to be prime numbers. We also make this choice in the presentation of our modular algorithms by considering prime numbers instead of arbitrary integers. These prime numbers are denoted by μ.

2.5.2 Modular gcd computation

We illustrate here how the tools from the previous section can be used on a classical problem, namely computing a gcd of two univariate polynomials with integer coefficients. As mentioned in the beginning of this section, the *CRT*-based approach for computing the gcd is particularly efficient since it eliminates the growth of intermediate coefficients that usually appears in the Euclidean-like algorithms.

Following [Yap 2000, §4], we first review some important properties of the gcd under transformations by modular homomorphisms. Then we present the general framework behind the modular gcd algorithm. We focus our attention only on the results we need for the description of the algorithms presented in this thesis and refer to [Yap 2000, §4.6] for a complete and a detailed overview.

Theorem 2.5.5. *Let* $P, Q \in \mathbb{Z}[X]$ *and* μ *be an integer such that* $\phi_\mu(Lc(P)) \neq 0$ *or* $\phi_\mu(Lc(Q)) \neq 0$*. Then,*

$$\deg(\gcd(\phi_\mu(P), \phi_\mu(Q))) \geqslant \deg(\gcd(P, Q)).$$

Proof. Writing $P = \gcd(P, Q) \times D$ and $Q = \gcd(P, Q) \times H$ and applying ϕ_μ to both sides of the two equations, we obtain $\phi_\mu(P) = \phi_\mu(\gcd(P, Q)) \times \phi_\mu(D)$ and $\phi_\mu(Q) = \phi_\mu(\gcd(P, Q)) \times \phi_\mu(H)$. The polynomial $\phi_\mu(\gcd(P, Q))$ is not zero in $\mathbb{Z}_\mu$ since $\phi_\mu(P) \not\equiv 0$ or $\phi_\mu(Q) \not\equiv 0$ and it divides both $\phi_\mu(P)$ and $\phi_\mu(Q)$, thus, it also divides $\gcd(\phi_\mu(P), \phi_\mu(Q))$, thus $\deg(\phi_\mu(\gcd(P, Q))) \leqslant \deg(\gcd(\phi_\mu(P), \phi_\mu(Q)))$. On the other hand, since $\gcd(P, Q)$ divides P and Q, then $Lc(\gcd(P, Q))$ divides $Lc(P)$ and $Lc(Q)$ in $\mathbb{Z}$. Thus, $\phi_\mu(Lc(P)) \not\equiv 0$ or $\phi_\mu(Lc(Q)) \neq 0$ implies that $\phi_\mu(Lc(\gcd(P, Q))) \neq 0$, thus $\deg(\phi_\mu(\gcd(P, Q))) = deg(\gcd(P, Q))$ and the claim follows. □

From the previous proof, one can notice that for a given prime number μ, if $\phi_\mu(Lc(P)) \neq 0$ or $\phi_\mu(Lc(Q)) \neq 0$ and $\deg(\gcd(\phi_\mu(P), \phi_\mu(Q))) = \deg(\gcd(P, Q))$ then, there exists $c \in \mathbb{Z}_\mu$ such that $\phi_\mu(\gcd(P, Q)) = c \times \gcd(\phi_\mu(P), \phi_\mu(Q))$. In addition, computing a normalized form of the gcd in $\mathbb{Q}[X]$ allows to turn the constant c to 1 and thus to obtain the equality $\phi_\mu(\gcd(P, Q)) = \gcd(\phi_\mu(P), \phi_\mu(Q))$ (see [Brown 1971]). Using the terminology of the previous section, the homomorphisms ϕ_μ for which $\phi_\mu(Lc(P)) = 0$ and $\phi_\mu(Lc(Q)) = 0$, or $\deg(\gcd(\phi_\mu(P), \phi_\mu(Q))) > \deg(\gcd(P, Q))$ are called *unlucky* homomorphisms. Clearly, using such homomorphisms for lifting the gcd will leads to a wrong result. The following theorem gives a more precise characterization of *unlucky* homomorphisms which yields an upper-bound on their number.

Theorem 2.5.6. *[Yap 2000, Lemma 4.11] Let $P, Q \in \mathbb{Z}[X]$ and μ be a prime such that $\phi_\mu(Lc(P)) \neq 0$ or $\phi_\mu(Lc(Q)) \neq 0$. ϕ_μ is an unlucky homomorphism for computing $\gcd(P, Q)$ if and only if $\phi_\mu(sres_d(P, Q)) = 0$ where d is the degree of $\gcd(P, Q)$.*

Theorem 2.5.7. *[Yap 2000, Lemma 4.12] Let $P, Q \in \mathbb{Z}[X]$ of degree at most d and maximum bitsize τ. If π is the product of all unlucky primes for computing the gcd of P and Q, then*

$$\tau_\pi \leqslant 2\tau(d + 1).$$

Using the two previous theorems, several algorithms for computing the gcd of two univariate polynomials using modular techniques have been presented (see for example [von zur Gathen 2003] for probabilistic algorithms and [Yap 2000] for deterministic ones). The following result gives the expected bit complexity of the Las-Vegas algorithm presented in [von zur Gathen 2003, §6.7].

Theorem 2.5.8. *[von zur Gathen 2003, Corollary 11.11] Let $P, Q \in \mathbb{Z}[x]$ of degree at most d and maximum bitsize τ. The gcd of P and Q can be computed using an expected number of $\widetilde{O}_B(d^2 + d\tau)$ bit operations.*

Chapter 3

Separating Linear Form

Contents

We address in this chapter the deterministic computation of a separating linear form of a system of bivariate polynomials with integer coefficients, that is a linear combination of the variables that takes different values when evaluated at distinct (complex) solutions of the system. In other words, a separating linear form defines a shear of the coordinate system that sends the algebraic system in generic position, in the sense that no two distinct solutions are vertically aligned. As mentioned in the introduction, the computation of such linear forms is at the core of most algorithms that solve algebraic systems by computing rational parametrizations of the solutions and, moreover, the computation of a separating linear form is the bottleneck of these algorithms, in terms of worst-case bit complexity.

The results presented in this chapter have been the subject of a publication in the ISSAC 2013 conference [Bouzidi 2013b].

3.1 Introduction

Let P and Q be two bivariate polynomials of total degree bounded by d and integer coefficients of maximum bitsize τ. Let $I = \langle P, Q\rangle$ be the ideal they define and suppose that I is zero-dimensional. The goal is to find a linear form $T = X + aY$, with $a \in \mathbb{Z}$, that separates the solutions of I. As discussed above and in the introduction (Chapter 1), computing such a linear form is a key step of many algorithms for solving algebraic systems through univariate representations.

We first outline a classical algorithm which is essentially the same as those proposed, for instance, in [Diochnos 2009, Lemma 16] and [Kerber 2012, Theorem 24][1] and whose complexity, in $\widetilde{O}_B(d^{10} + d^9\tau)$, is the best known so far for this problem. This algorithm serves two purposes: it gives some insight on the more involved $\widetilde{O}_B(d^8 + d^7\tau)$-time algorithm that follows and it will be used in that algorithm but over $\mathbb{Z}/\mu\mathbb{Z}$ instead of $\mathbb{Z}$.

Known $\widetilde{\mathbf{O}}_\mathbf{B}(\mathbf{d}^{10} + \mathbf{d}^9\tau)$-time algorithm for computing a separating linear form. The idea is to work with a "generic" linear form $T = X + SY$, where S is an indeterminate, and find conditions such that the specialization of S by an integer a gives a separating form. We thus consider $P(T - SY, Y)$ and $Q(T - SY, Y)$, the "generic" sheared polynomials associated to P and Q, and $R(T, S)$ their resultant with respect to Y. This polynomial has been extensively used and defined in several context; see for instance the related u-resultant [Van der Waerden 1930].

It is known that, in a set $\mathcal{S}$ of d^4 integers, there exists at least one integer a such that $X + aY$ is a separating form for I since I has at most d^2 solutions which define at most $\binom{d^2}{2}$ directions in which two solutions are aligned. Hence, a separating form can be found by computing, for every a in $\mathcal{S}$, the degree of the squarefree part of $R(T, a)$ and by choosing one a for which this degree is maximum. Indeed, for any (possibly non-separating) linear form $X + aY$, the number of distinct roots of $R(T, a)$, which is the degree of its squarefree part, is always smaller than or equal to the number of distinct solutions of I, and equality is attained when the linear form $X + aY$ is separating (Lemma 3.3.3). The complexity of this algorithm is in $\widetilde{O}_B(d^{10} + d^9\tau)$ because, for d^4 values of a, the polynomial $R(T, a)$ can be shown to be of degree $O(d^2)$ and bitsize $\widetilde{O}(d^2 + d\tau)$, and its squarefree part can be computed in $\widetilde{O}_B(d^6 + d^5\tau)$ time.

We now outline the algorithm that we present in this chapter.

$\widetilde{\mathbf{O}}_\mathbf{B}(\mathbf{d}^8 + \mathbf{d}^7\tau)$-time algorithm for computing a separating linear form. To reduce the complexity of the search for a separating form, one can first consider to perform naively the above algorithm on the system $I_\mu =$

---

[1] The stated complexity of [Kerber 2012, Theorem 24] is $\widetilde{O}_B(d^9\tau)$, but it seems the fact that the sheared polynomials have bitsize in $\widetilde{O}(d + \tau)$ (see Lemma 3.2.1) instead of $\widetilde{O}(\tau)$ has been overlooked in their proof.

$\langle P \bmod \mu, Q \bmod \mu\rangle$ in $\mathbb{Z}_\mu = \mathbb{Z}/\mu\mathbb{Z}$, where μ is a prime number upper bounded by some polynomial in d and τ (so that the bit complexity of arithmetic operations in $\mathbb{Z}_\mu$ is polylogarithmic in d and τ). The resultant $R_\mu(T,S)$ of $P(X-SY,Y) \bmod \mu$ and $Q(X-SY,Y) \bmod \mu$ with respect to Y can be computed in $\widetilde{O}_B(d^6+d^5\tau)$ bit operations and, since its degree is at most $2d^2$ in each variable, evaluating it at $S=a$ in $\mathbb{Z}_\mu$ can be easily done in $\widetilde{O}_B(d^4)$ bit operations. Then, the computation of its squarefree part does not suffer anymore from the coefficient growth, and it becomes softly linear in its degree, that is $\widetilde{O}_B(d^2)$. Considering d^4 choices of a, we get an algorithm that computes a separating form for I_μ in $\widetilde{O}_B(d^8)$ time in $\mathbb{Z}_\mu$. However, a serious problem remains, that is to ensure that a separating form for I_μ is also a separating form for I. This issue requires to develop a more subtle algorithm.

We first show, in Section 3.3, a critical property (Proposition 3.3.2) which states that a separating linear form over $\mathbb{Z}_\mu$ is also separating over $\mathbb{Z}$ when μ is a *lucky* prime number, which is, essentially, a prime such that the number of solutions of $\langle P,Q\rangle$ is the same over $\mathbb{Z}$ and over $\mathbb{Z}_\mu$. We then show in Sections 3.4 to 3.6 how to compute such a lucky prime number. We do that by first proving in Section 3.4 that, under mild conditions on μ, the number of solutions of I_μ is always less than or equal to the number of solutions of I (Proposition 3.4.1) and then by computing a bound on the number of unlucky primes (Proposition 3.4.2). Computing a lucky prime can then be done by choosing a μ that maximizes the number of solutions of I_μ among a set of primes of cardinality $\widetilde{O}(d^4+d^3\tau)$. For that purpose, we present in Section 3.5 a new algorithm, of independent interest, for computing in $\widetilde{O}(d^4)$ arithmetic operations in $\mathbb{Z}_\mu$ the number of distinct solutions of the system I_μ; this algorithm is based on a classical triangular decomposition. This yields, in Section 3.6, a $\widetilde{O}_B(d^8+d^7\tau)$-time algorithm for computing a lucky prime μ in $\widetilde{O}(d^4+d^3\tau)$. Now, μ is fixed, and we can apply the algorithm outlined above for computing a separating form for I_μ in $\mathbb{Z}_\mu$ in $\widetilde{O}_B(d^8)$ time (Section 3.7). This form, which is also separating for I, is thus obtained with a total bit complexity of $\widetilde{O}_B(d^8+d^7\tau)$ (Theorem 3.7.1).

3.2 Notation and preliminaries

We introduce the following notation which are extensively used throughout this Chapter and the next one. Given the two input polynomials P and Q, we consider the "generic" change of variables $X = T - SY$, and define the "sheared" polynomials $P(T-SY,Y)$, $Q(T-SY,Y)$, and their resultant with respect to Y,

$$R(T,S) = Res_Y(P(T-SY,Y), Q(T-SY,Y)). \tag{3.1}$$

The complexity bounds on the degree, bitsize and computation of these polynomials are analyzed at the end of this section in Lemma 3.2.1. Let $L_R(S)$ be the leading coefficient of $R(T,S)$ seen as a polynomial in T. Let $L_P(S)$ and $L_Q(S)$ be the leading coefficients of $P(T-SY,Y)$ and $Q(T-SY,Y)$, seen as polynomials in Y; it is straightforward that these leading coefficients do not depend on T. In other words:

$$\begin{gathered} L_P(S) = Lc_Y(P(T-SY,Y)), L_Q(S) = Lc_Y(Q(T-SY,Y)), \\ L_R(S) = Lc_T(R(T,S)). \end{gathered} \tag{3.2}$$

Lemma 3.2.1. *Let P and Q in $\mathbb{Z}[X,Y]$ be of total degree at most d and maximum bitsize τ. The sheared polynomials $P(T-SY,Y)$ and $Q(T-SY,Y)$ can be expanded in $\widetilde{O}_B(d^4+d^3\tau)$ and their bitsizes are in $\widetilde{O}(d+\tau)$. The resultant $R(T,S)$ can be computed in $\widetilde{O}_B(d^7+d^6\tau)$ bit operations and $\widetilde{O}(d^5)$ arithmetic operations in $\mathbb{Z}$; its degree is at most $2d^2$ in each variable and its bitsize is in $\widetilde{O}(d^2+d\tau)$.*

Proof. Writing P as $\sum_{i=0}^{d} p_i(Y)X^i$, expending the substitution of X by $T-SY$ needs the computation of the successive powers $(T-SY)^i$ for i from 1 to d. The binomial formula shows that each polynomial $(T-SY)^i$ is the sum of $i+1$ monomials, with coefficients of bitsize in $O(i\log i)$. Using the recursion formula $(T-SY)^i = (T-SY)^{i-1}(T-SY)$, given the polynomial $(T-SY)^{i-1}$, the computation of $(T-SY)^i$ requires $2i$ multiplications of coefficients having bitsize in $O(i\log i)$, which can be done in $\widetilde{O}_B(i^2\log i)$ bit operations. The complexity of computing all the powers is thus in $\widetilde{O}_B(d^3\log d)$. The second step is to multiply $p_i(Y)$ by $(T-SY)^i$ for $i=1,\ldots,d$. Each polynomial multiplication can be done with $O(d^2)$ multiplications of integers of bitsize in $O(\tau)$ or in $O(d\log d)$, and thus it can be done in $\widetilde{O}_B(d^2(\tau+d\log d))$ bit operations and yields polynomials of bitsize $O(\tau+d\log d)$. For the d multiplications the total cost is in $\widetilde{O}_B(d^3(\tau+d\log d))$. Consequently the computation of $P(T-SY,Y)$ and $Q(T-SY,Y)$ can be done in $\widetilde{O}_B(d^3(\tau+d))$ bit operations and these polynomials have bitsize in $\widetilde{O}(\tau+d)$. In addition, since $P(T-SY,Y)$ and $Q(T-SY,Y)$ are trivariate polynomials of partial degree in all variables bounded by d, Theorem 2.4.16 implies the claims on $R(T,S)$. □

Throughout this chapter, we assume that the two input polynomials P and Q are coprime in $\mathbb{Z}[X,Y]$, that they define the ideal I, that their maximum total degree d is at least 2 and that their coefficients have maximum bitsize τ. Note that the coprimality of P and Q is implicitly tested during Algorithm 5 because they are coprime if and only if $R(T,S)$ does not identically vanish. By abuse of notation, some complexity $\widetilde{O}_B(d^k)$ may refer to a complexity in which polylogarithmic factors in d *and* in τ are omitted. $I_\mu = \langle P_\mu, Q_\mu\rangle$

denotes the ideal generated by $P_\mu = \phi_\mu(P)$ and $Q_\mu = \phi_\mu(Q)$. Similarly as in Equation (3.1), we define $R_\mu(T,S)$ as the resultant of $P_\mu(T-SY,Y)$ and $Q_\mu(T-SY,Y)$ with respect to Y, and we define $L_{P_\mu}(S)$, $L_{Q_\mu}(S)$, and $L_{R_\mu}(S)$, similarly as in (3.2). We refer to the overview in Section 3.1 for the organization of this chapter.

3.3 Separating linear form over $\mathbb{Z}_\mu$ versus $\mathbb{Z}$

We first introduce the notion of lucky prime numbers μ which are, roughly speaking, primes μ for which the number of distinct solutions of $\langle P,Q\rangle$ does not change when considering the polynomials modulo μ. We then show the critical property that, if a linear form is separating modulo such a μ, then it is also separating over $\mathbb{Z}$.

Definition 3.3.1. *A prime number μ is said to be* **lucky** *for an ideal $I = \langle P,Q\rangle$ if it is larger than $2d^4$ and satisfies*

$$\phi_\mu(L_P(S))\ \phi_\mu(L_Q(S))\ \phi_\mu(L_R(S)) \not\equiv 0 \quad \textit{and} \quad \#V(I) = \#V(I_\mu).$$

Proposition 3.3.2. *Let μ be a lucky prime for the ideal $I = \langle P,Q\rangle$ and let $a < \mu$ be an integer*[2] *such that*

$$\phi_\mu(L_P(a))\ \phi_\mu(L_Q(a))\ \phi_\mu(L_R(a)) \neq 0.$$

If $X + aY$ separates $V(I_\mu)$, it also separates $V(I)$.

The key idea of the proof of Proposition 3.3.2, as well as Propositions 3.4.1 and 3.4.2, is to prove the following inequalities (under the hypothesis that various leading terms do not vanish)

$$\#V(I_\mu) \geqslant d_T(\overline{R_\mu(T,a)}) \leqslant d_T(\overline{R(T,a)}) \leqslant \#V(I) \tag{3.3}$$

and argue that the first (resp. last) one is an equality if $X + aY$ separates $V(I_\mu)$ (resp. $V(I)$), and that the middle one is an equality except for finitely many μ. We establish these claims in Lemmas 3.3.3 and 3.3.5. As mentioned in Section 3.1, Lemma 3.3.3 is the key property in the classical algorithm for computing a separating form for I, which algorithm we will use over $\mathbb{Z}_\mu$ to compute a separating form for I_μ in Section 3.7. For completeness, we outline its proof (see also [Diochnos 2009, Lemma 16] or [Basu 2006, Proposition 11.23]). Recall that P and Q are assumed to be coprime but not P_μ and Q_μ; we address this issue in Lemma 3.3.4.

---

[2] We assume $a < \mu$ for clarity so that the linear form $X + aY$ is “identical” in $\mathbb{Z}$ and in $\mathbb{Z}_\mu$. This hypothesis is however not needed and we actually prove that if $X + \phi_\mu(a)Y$ separates $V(I_\mu)$, then $X + aY$ separates $V(I)$.

Lemma 3.3.3. *If $a \in \mathbb{Z}$ is such that $L_P(a)\, L_Q(a) \neq 0$ then $d_T(\overline{R(T,a)}) \leqslant \#V(I)$ and they are equal if and only if $X + aY$ separates $V(I)$. The same holds over $\mathbb{Z}_\mu$, that is for P_μ, Q_μ, R_μ and I_μ, provided P_μ and Q_μ are coprime.*

Proof. Since $L_P(a)\ L_Q(a) \neq 0$, the resultant $R(T,S)$ can be specialized at $S = a$, that is $R(T,a) = Res_Y(P(T-aY,Y), Q(T-aY,Y))$. On the other hand, the sheared polynomials $P(T-aY,Y)$ and $Q(T-aY,Y)$ are coprime (since P and Q are coprime) and since $L_P(a)\ L_Q(a) \neq 0$, they have no common solution at infinity in the Y-direction. Thus, according to Proposition 2.4.5 the roots of their resultant with respect to Y are the T-coordinates of the (affine) solutions of $I_a = \langle P(T-aY,Y), Q(T-aY,Y)\rangle$. Hence, $d_T(\overline{R(T,a)}) \leqslant \#V(I_a) = \#V(I)$. Moreover, if $X + aY$ separates $V(I)$, $T = X + aY$ takes distinct values for every solution in $V(I)$, and since these values of T are roots of $R(T,a)$, $d_T(\overline{R(T,a)}) \geqslant \#V(I)$ and thus they are equal. Conversely, if $d_T(\overline{R(T,a)}) = \#V(I)$, $R(T,a)$ admits $\#V(I)$ distinct roots $T = X + aY$ which means that $X + aY$ separates all the solutions of $V(I)$. The same argument holds over $\mathbb{Z}_\mu$. □

The following two lemmas state rather standard properties. For completeness and readers' convenience, we provide proofs of these statements for which we could not find accurate references.

Lemma 3.3.4. *If $\phi_\mu(L_P(S))\ \phi_\mu(L_Q(S))\ \phi_\mu(L_R(S)) \not\equiv 0$ and $\mu > 4d^2$ then P_μ and Q_μ are coprime in $\mathbb{Z}_\mu[X,Y]$.*

Proof. Since $\phi_\mu(L_P(S))\ \phi_\mu(L_Q(S)) \not\equiv 0$, the property of specialization of resultants (Theorem 2.4.4) yields that $\phi_\mu(R(T,S)) = R_\mu(T,S)$ and $\phi_\mu(L_R(S)) \not\equiv 0$ implies that $R_\mu(T,S) \not\equiv 0$. We can thus choose a value $S = a \in \mathbb{Z}_\mu$ so that $R_\mu(T,a) \not\equiv 0$ and $L_{P_\mu}(a)\, L_{Q_\mu}(a) \neq 0$; indeed, $\mu > 4d^2$ and $\phi_\mu(L_R(S))$, $L_{P_\mu}(S)$ and $L_{Q_\mu}(S)$ have degree at most $2d^2$, d and d respectively (Theorem 2.4.16). For such a value, the resultant of $P_\mu(T-aY,Y)$ and $Q_\mu(T-aY,Y)$ is $R_\mu(T,a)$. This resultant is not identically zero, the leading coefficients (in Y) $L_{P_\mu}(a)$ and $L_{Q_\mu}(a)$ do not depend on T (see Eq. (3.2)) and are not zero, thus $P_\mu(T-aY,Y)$ and $Q_\mu(T-aY,Y)$ are coprime. The result follows. □

Lemma 3.3.5. *Let μ be a prime and a be an integer such that $\phi_\mu(L_P(a))\ \phi_\mu(L_Q(a))\ \phi_\mu(L_R(a)) \neq 0$, then $d_T(\overline{R_\mu(T,a)}) \leqslant d_T(\overline{R(T,a)})$.*

Proof. We first observe that the degree of $R(T,a)$ and $\phi_\mu(R(T,a))$ are the same. Indeed, $\phi_\mu(L_R(a)) \neq 0$ by hypothesis and thus $L_R(a) \neq 0$. Thus, the leading coefficient $L_R(S)$ of $R(T,S)$ does not vanish when specialized at $S = a$, and it also does not vanish when furthermore taken modulo μ.

Now, the degree of the squarefree part of a univariate polynomial is its degree minus the degree of its gcd with its derivative. Furthermore, according to Theorem 2.5.5, the degree of the gcd of two univariate polynomials cannot decrease by reduction modulo μ, if their leading coefficients do not both vanish modulo μ. The leading coefficients of $R(T,a)$ and its derivative do not vanish modulo μ since $\phi_\mu(L_R(a)) \neq 0$, and thus

$$\begin{aligned} d_T(\overline{R(T,a)}) &= d_T(R(T,a)) - d_T(\gcd(R(T,a), R'(T,a))) \\ &\geqslant d_T(\phi_\mu(R(T,a))) - d_T(\gcd(\phi_\mu(R(T,a)), \phi_\mu(R'(T,a)))) \\ &= d_T(\overline{\phi_\mu(R(T,a))}). \end{aligned}$$

We finally argue that $\phi_\mu(R(T,a)) = R_\mu(T,a)$. By hypothesis, $\phi_\mu(L_P(S))$ and $\phi_\mu(L_Q(S))$ do not identically vanish, thus we can specialize the resultant R by ϕ_μ, that is $\phi_\mu(R(T,S)) = Res_Y(\phi_\mu(P(T-SY,Y)), \phi_\mu(Q(T-SY,Y)))$ (Theorem 2.4.4). Hence, $\phi_\mu(R(T,S)) = R_\mu(T,S)$. The evaluation at $S=a$ and the reduction modulo μ commute (in $\mathbb{Z}_\mu$), thus $\phi_\mu(R(T,a)) = R_\mu(T,a)$ in $\mathbb{Z}_\mu[T]$, which concludes the proof of the lemma. ☐

Proof of Proposition 3.3.2. By Lemmas 3.3.3, 3.3.4 and 3.3.5, if μ is a prime and a is an integer such that $X + aY$ separates $V(I_\mu)$ and $\phi_\mu(L_P(a))\ \phi_\mu(L_Q(a))\ \phi_\mu(L_R(a)) \neq 0$, then

$$\#V(I_\mu) = d_T(\overline{R_\mu(T,a)}) \leqslant d_T(\overline{R(T,a)}) \leqslant \#V(I).$$

Since μ is lucky, $\#V(I_\mu) = \#V(I)$ thus $d_T(\overline{R(T,a)}) = \#V(I)$ and by Lemma 3.3.3, $X + aY$ separates $V(I)$. ☐

3.4 Number of solutions of I_μ versus I

As shown in Proposition 3.3.2, the knowledge of a lucky prime permits to search for separating linear forms over $\mathbb{Z}_\mu$ rather than over $\mathbb{Z}$. We prove here two propositions that are critical for computing a lucky prime, which state that the number of solutions of $I_\mu = \langle P_\mu, Q_\mu \rangle$ is always at most that of $I = \langle P, Q \rangle$ and give a bound on the number of unlucky primes.

Proposition 3.4.1. *Let $I = \langle P, Q \rangle$ be a zero-dimensional ideal in $\mathbb{Z}[X,Y]$. If a prime μ is larger than $2d^4$ and*

$$\phi_\mu(L_P(S))\ \phi_\mu(L_Q(S))\ \phi_\mu(L_R(S)) \not\equiv 0$$

then $\#V(I_\mu) \leqslant \#V(I)$.

Proof. Let μ be a prime that satisfies the hypotheses of the proposition. We also consider an integer $a < \mu$ such that $\phi_\mu(L_P(a))\ \phi_\mu(L_Q(a))\ \phi_\mu(L_R(a)) \neq 0$ and such that the linear form $X + aY$ is separating for I_μ. Such an integer exists because (i) $\phi_\mu(L_P(S))$, $\phi_\mu(L_Q(S))$, and $\phi_\mu(L_R(S))$ are not identically zero by hypothesis and they have degree at most d or $2d^2$ (Theorem 2.4.16) and, as mentioned earlier, (ii) I_μ is zero dimensional (Lemma 3.3.4) and it has at most d^2 solutions which define at most $\binom{d^2}{2}$ directions in which two solutions are aligned. Since $2d + 2d^2 + \binom{d^2}{2} < 2d^4$ (for $d \geqslant 2$), there exists such an integer $a \leqslant 2d^4 < \mu$. With such an a, we can apply Lemmas 3.3.3 and 3.3.5 which imply that $\#V(I_\mu) = d_T(\overline{R_\mu(T,a)}) \leqslant d_T(\overline{R(T,a)}) \leqslant \#V(I)$. □

Next, we bound the number of primes that are unlucky for the ideal $\langle P, Q\rangle$.

Proposition 3.4.2. *An upper bound on the number of unlucky primes for the ideal $\langle P, Q\rangle$ can be explicitly computed in terms of d and τ, and this bound is in $\widetilde{O}(d^4 + d^3\tau)$.*

Proof. According to Definition 3.3.1, a prime μ is unlucky if it is smaller than $2d^4$, if $\phi_\mu(L_P(S)L_Q(S)\ L_R(S)) = 0$, or if $\#V(I) \neq \#V(I_\mu)$. In the following, we consider $\mu > 2d^4$. We first determine some conditions on μ that ensure that $\#V(I) = \#V(I_\mu)$, and we then bound the number of μ that do not satisfy these conditions. As we will see, under these conditions, $L_P(S)$, $L_Q(S)$, and $L_R(S)$ do not vanish modulo μ and thus this constraint is redundant.

The first part of the proof is similar in spirit to that of Proposition 3.4.1 in which we first fixed a prime μ and then specialized the polynomials at $S = a$ such that the form $X + aY$ was separating for I_μ. Here, we first choose a such that $X + aY$ is separating for I. With some conditions on μ, Lemmas 3.3.3 and 3.3.5 imply Equation (3.4) and we determine some more conditions on μ such that the middle inequality of (3.4) is an equality. We thus get $\#V(I_\mu) \geqslant \#V(I)$ which is the converse of that of Proposition 3.4.1 and thus $\#V(I_\mu) = \#V(I)$. In the second part of the proof, we bound the number of μ that violate the conditions we considered.

Prime numbers such that $\#V(I) \neq \#V(I_\mu)$. Let a be such that the form $X + aY$ separates $V(I)$ and $L_P(a)\, L_Q(a)\, L_R(a) \neq 0$.[3] Similarly as in the proof of Proposition 3.4.1, we can choose $a \leqslant 2d^4$.

We consider any prime μ such that $\phi_\mu(L_P(a))\ \phi_\mu(L_Q(a))\ \phi_\mu(L_R(a)) \neq 0$, so that we can apply Lemmas 3.3.3 and 3.3.5. Since $X + aY$ separates $V(I)$, these lemmas yield that

$$\#V(I_\mu) \geqslant d_T(\overline{R_\mu(T,a)}) \leqslant d_T(\overline{R(T,a)}) = \#V(I). \tag{3.4}$$

Now, $d_T(\overline{R(T,a)}) = d_T(R(T,a)) - d_T(\gcd(R(T,a), R'(T,a)))$, and similarly for $R_\mu(T,a)$. The leading coefficient of $R(T,S)$ with respect to T is $L_R(S)$, and

---

[3] It can be shown that $L_P(a)\, L_Q(a) \neq 0$ implies $L_R(a) \neq 0$ (see for instance Lemma 4.2.5) but this property does not simplify the proof.

since it does not vanish at $S = a$, $L_R(a)$ is the leading coefficient of $R(T, a)$. In addition, we have shown in the proof of Lemma 3.3.5 that $R_\mu(T, a) = \phi_\mu(R(T, a))$, hence the hypothesis $\phi_\mu(L_R(a)) \neq 0$ implies that $R_\mu(T, a)$ and $R(T, a)$ have the same degree. It follows that, *if* μ *is such that the degree of* $\gcd(R(T, a), R'(T, a))$ *does not change when* $R(T, a)$ *and* $R'(T, a)$ *are reduced modulo* μ, we have

$$\#V(I_\mu) \geqslant d_T(\overline{R_\mu(T, a)}) = d_T(\overline{R(T, a)}) = \#V(I).$$

Since $\phi_\mu(L_P(a))\ \phi_\mu(L_Q(a))\ \phi_\mu(L_R(a)) \neq 0$, we can apply Proposition 3.4.1 which yields that $\#V(I_\mu) \leqslant \#V(I)$ and thus $\#V(I_\mu) = \#V(I)$.

Therefore, the primes μ such that $\#V(I_\mu) \neq \#V(I)$ are among those such that $L_P(a)$, $L_Q(a)$ or $L_R(a)$ vanishes modulo μ or such that the degree of $\gcd(R(T, a), R'(T, a))$ changes when $R(T, a)$ and $R'(T, a)$ are reduced modulo μ. Note that if $L_P(a)$, $L_Q(a)$, and $L_R(a)$ do not vanish modulo μ, then $L_P(S)$, $L_Q(S)$, and $L_R(S)$ do not identically vanish modulo μ.

Bounding the number of prime divisors of $L_P(a)$, $L_Q(a)$ *or* $L_R(a)$. The number of prime divisors of an integer z is bounded by its bitsize. Indeed, its bitsize is $\lfloor \log z \rfloor + 1$ and its factorization into w (possibly identical) prime numbers directly yields that $2^w \leqslant \prod_{i=1}^{w} z_i = z = 2^{\log z} \leqslant 2^{\lfloor \log z \rfloor + 1}$. We can thus bound the number of prime divisors by bounding the bitsize of $L_P(a)$, $L_Q(a)$ and $L_R(a)$. We start by bounding the bitsize of $L_P(S)$, $L_Q(S)$ and $L_R(S)$.

Each coefficient of $P(T - SY, Y)$ has bitsize at most $\tau' = \tau + d \log d + \log(d+1) + 1$. Indeed, $(T - SY)^i$ is a sum of $i+1$ monomials whose coefficients are binomials $\binom{i \leqslant d}{j} < d^d$. The claim follows since each coefficient of $P(T - SY, Y)$ is the sum of at most $d + 1$ such binomials, each multiplied by a coefficient of $P(X, Y)$ which has bitsize at most τ. We get the same bound for the coefficients of $Q(T - SY, Y)$ and thus for $L_P(S)$ and $L_Q(S)$ as well. Concerning $L_R(S)$, we have that $R(T, S)$ is the resultant of $P(T - SY, Y)$ and $Q(T - SY, Y)$ thus, by Theorem 2.4.16, its coefficients are of bitsize $\widetilde{O}(d\tau')$. In fact, an upper bound can be explicitly computed using, for instance, the bound of [Basu 2006, Theorem 8.46] which implies that the resultant of two trivariate polynomials of total degree d' and bitsize τ' has bitsize at most $2d'(\tau' + \lfloor \log 2d' \rfloor + 1) + 2(\lfloor \log(2d'^2 + 1) \rfloor + 1)$, which is in $\widetilde{O}(d^2 + d\tau)$ in our case. Therefore, $L_P(S)$, $L_Q(S)$ and $L_R(S)$ have degree at most $2d^2$ and their bitsizes can be explicitly bounded by a function of d and τ in $\widetilde{O}(d^2 + d\tau)$.

Finally, since $a \leqslant 2d^4$, its bitsize is at most $\sigma = 4 \log d + 2$. It is straightforward that the result of an evaluation of a univariate polynomial of degree at most d' and bitsize τ' at an integer value of bitsize σ has bitsize at most $d'\sigma + \tau' + \log(d' + 1) + 1$. Here $d' \leqslant 2d^2$ and τ' is in $\widetilde{O}(d^2 + d\tau)$. We thus proved that we can compute an explicit bound, in $\widetilde{O}(d^2 + d\tau)$, on the number of prime divisors of $L_P(a)$, $L_Q(a)$, or $L_R(a)$.

Bounding the number of prime μ such that the degree of $\gcd(R(T,a), R'(T,a))$ *changes when $R(T,a)$ and $R'(T,a)$ are reduced modulo μ.* By Theorem 2.5.7, given two univariate polynomials in $\mathbb{Z}[X]$ of degree at most d' and bitsize at most τ', the product of all μ, such that the degree of the gcd of the two polynomials changes when the polynomials are considered modulo μ, is bounded by $(2^{\tau'}\sqrt{d'+1})^{2d'+2}$. As noted above, the number of such primes μ is bounded by the bitsize of this bound, and thus is bounded by $(d'+1)\,(2\tau'+\log(d'+1))+1$. Here $d' \leqslant 2d^2$ and τ' is in $\widetilde{O}(d^2+d\tau)$ since our explicit bound on the bitsize of $L_R(a)$ holds as well for the bitsize of $R(T,a)$, and, since $R(T,a)$ is of degree at most $2d^2$, the bitsize of $R'(T,a)$ is bounded by that of $R(T,a)$ plus $1+\log 2d^2$. We thus obtain an explicit bound in $\widetilde{O}(d^4+d^3\tau)$ on the number of primes μ such that the degree of $\gcd(R(T,a), R'(T,a))$ changes when $R(T,a)$ and $R'(T,a)$ are reduced modulo μ.

The result follows by summing this bound with the bounds we obtained on the number of prime divisors of $L_P(a)$, $L_Q(a)$, or $L_R(a)$, and a bound (e.g. $2d^4$) on the number of primes smaller than $2d^4$. □

3.5 Counting the number of solutions of I_μ

For counting the number of (distinct) solutions of $I_\mu = \langle P_\mu, Q_\mu\rangle$, we use a classical algorithm for computing a triangular decomposition of an ideal defined by two bivariate polynomials. We first recall this algorithm, slightly adapted to our needs, and analyze its arithmetic complexity.

3.5.1 Triangular decomposition

Let P and Q be two polynomials in $\mathbb{F}[X,Y]$. A decomposition of the solutions of the system $\{P,Q\}$ using the subresultant sequence appears in the theory of triangular sets [Lazard 1991, Li 2011] and for the computation of topology of curves [Gonzalez-Vega 1996].

The idea is to use Theorem 2.4.15 which states that, after specialization at $X=\alpha$, the first (with respect to increasing i) nonzero subresultant $Sres_{Y,i}(P,Q)(\alpha,Y)$ is of degree i and is equal to the gcd of $P(\alpha,Y)$ and $Q(\alpha,Y)$. This induces a decomposition of the system $\{P,Q\}$ into triangular subsystems $(\{A_i(X), Sres_{Y,i}(P,Q)(X,Y)\})$ where a solution α of $A_i(X)=0$ is such that the system $\{P(\alpha,Y), Q(\alpha,Y)\}$ admits exactly i roots (counted with multiplicity), which are exactly those of $Sres_{Y,i}(P,Q)(\alpha,Y)$. Furthermore, these triangular subsystems are regular chains, i.e., the leading coefficient of the bivariate polynomial (seen in Y) is coprime with the univariate polynomial. For clarity and self-containedness, we recall this decomposition

Algorithm 2 Triangular decomposition [Gonzalez-Vega 1996, Li 2011]

```
Require: $P, Q$ in $\mathbb{F}[X,Y]$ coprime such that $Lc_Y(P)$ and $Lc_Y(Q)$ are coprime,[4] $d_Y(Q) \leqslant d_Y(P)$, and
        $A \in \mathbb{F}[X]$ squarefree.
Ensure: Triangular decomposition $\{(A_i(X),\ B_i(X,Y))\}_{i\in\mathcal{I}}$ such that $V(\langle P,Q,A\rangle)$ is the disjoint union of the sets $V(\langle A_i(X), B_i(X,Y)\rangle)_{i\in\mathcal{I}}$
Compute the subresultant sequence of $P$ and $Q$ with respect to $Y$: $B_i = Sres_{Y,i}(P,Q)$
$G_0 = \gcd(\overline{Res_Y(P,Q)}, A)$ and $\mathcal{T} = \emptyset$
for $i = 1$ to $d_Y(Q)$ do
    $G_i = \gcd(G_{i-1}, sres_{Y,i}(P,Q))$
    $A_i = G_{i-1}/G_i$
    if $d_X(A_i) > 0$, add $(A_i, B_i)$ to $\mathcal{T}$
end for
return $\mathcal{T} = \{(A_i(X), B_i(X,Y))\}_{i\in\mathcal{I}}$
```

in Algorithm 2, where, in addition, we restrict the solutions of the system $\{P,Q\}$ to those where some univariate polynomials $A(X)$ vanishes (A could be identically zero).

The following lemma states the correctness of Algorithm 2 which follows from Theorem 2.4.15 and from the fact that the solutions of P and Q project on the roots of their resultant (Proposition 2.4.5).

Lemma 3.5.1 ([Gonzalez-Vega 1996, Li 2011])**.** *Algorithm 2 computes a triangular decomposition* $\{(A_i(X),\ B_i(X,Y))\}_{i\in\mathcal{I}}$ *such that*

(i) *the set* $V(\langle P,Q,A\rangle)$ *is the disjoint union of the sets* $V(\langle A_i(X), B_i(X,Y)\rangle)_{i\in\mathcal{I}}$*,*

(ii) $\prod_{i\in\mathcal{I}} A_i$ *is squarefree,*

(iii) $\forall\alpha \in V(A_i)$*,* $B_i(\alpha, Y)$ *is of degree* i *and is equal to* $\gcd(P(\alpha,Y), Q(\alpha,Y))$*, and*

(iv) $A_i(X)$ *and* $Lc_Y(B_i(X,Y))$ *are coprime.*

In the following lemma, we analyze the complexity of Algorithm 2 for P and Q of degree at most d_X in X and d_Y in Y and A of degree at most d^2, where d denotes a bound on the total degree of P and Q. We will use Algorithm 2 with polynomials with coefficients in $\mathbb{F} = \mathbb{Z}_\mu$ and we thus only consider its arithmetic complexity in $\mathbb{F}$. Note that the bit complexity of this algorithm, over $\mathbb{Z}$, is analyzed in [Diochnos 2009, Theorem 19] and its

---

[4] The hypothesis that $Lc_Y(P)$ and $Lc_Y(Q)$ are coprime can be relaxed by applying the algorithm recursively (see Algorithm 7 for details).

arithmetic complexity is thus implicitly analyzed as well; for clarity, we provide here a short proof.

Lemma 3.5.2. *Algorithm 2 performs* $\widetilde{O}(d_X d_Y^3) = \widetilde{O}(d^4)$ *arithmetic operations in* $\mathbb{F}$.

Proof. From Theorem 2.4.17 (note that this lemma is stated for the coefficient ring $\mathbb{Z}$, but the arithmetic complexity is the same for any field $\mathbb{F}$), the subresultant sequence of P and Q can be computed in $\widetilde{O}(d_X d_Y^3)$ arithmetic operations, and the resultant as well as the principal subresultant coefficients have degrees in $O(d_X d_Y)$. The algorithm performs at most d_Y gcd computations between these univariate polynomials. The arithmetic complexity of one such gcd computation is softly linear in their degrees, that is $\widetilde{O}(d_X d_Y)$ (Theorem 2.3.8). Hence the arithmetic complexity of computing the systems $\{S_i\}_{i=1\dots d}$ is $\widetilde{O}(d_X d_Y^2)$. The total complexity of the triangular decomposition is hence dominated by the cost of the subresultant computation, that is $\widetilde{O}(d_X d_Y^3) = \widetilde{O}(d^4)$. ☐

3.5.2 Counting the number of solutions of I_μ

Algorithm 3 computes the number of distinct solutions of an ideal $I_\mu = \langle P_\mu, Q_\mu \rangle$ of $\mathbb{Z}_\mu[X, Y]$. Roughly speaking, this algorithm first performs one triangular decomposition with the input polynomials P_μ and Q_μ, and then performs a sequence of triangular decompositions with polynomials resulting from this decomposition. The result is close to a radical triangular decomposition and the number of solutions of I_μ can be read, with a simple formula, from the degrees of the polynomials in the decomposition. Note that Algorithm 3, as Algorithm 2, is valid for any base field $\mathbb{F}$ but, since we will only use it over $\mathbb{Z}_\mu$, we state it and analyze its complexity in this case.

Lemma 3.5.3. *Algorithm 3 computes the number of distinct solutions of* $\langle P_\mu, Q_\mu \rangle$.

Proof. The shear of Line 1 allows to fulfill the requirement of the triangular decomposition algorithm, called in Line 2, that the input polynomials have coprime leading coefficients. Once the generically sheared polynomial $P_\mu(X - SY, Y)$ is computed (in $\mathbb{Z}_\mu[S, X, Y]$), a specific shear value $b \in \mathbb{Z}_\mu$ can be selected by evaluating the univariate polynomial $L_{P_\mu}(S) = Lc_Y(P_\mu(X - SY, Y))$ at $d+1$ elements of $\mathbb{Z}_\mu$. The polynomial does not vanish at one of these values since it is of degree at most d and $d < \mu$. Note that such a shear clearly does not change the number of solutions.

According to Lemma 3.5.1, the triangular decomposition $\{(A_i(X), B_i(X, Y))\}_{i \in \mathcal{I}}$ computed in Line 2 is such that the solutions

Algorithm 3 Number of distinct solutions of $\langle P_\mu, Q_\mu\rangle$

```
Require: $P_\mu, Q_\mu$ in $\mathbb{Z}_\mu[X,Y]$ coprime, $\mu$ larger than their total degree
Ensure: Number of distinct solutions of $\langle P_\mu, Q_\mu\rangle$
Shear $P_\mu$ and $Q_\mu$ by replacing $X$ by $X - bY$ with $b \in \mathbb{Z}_\mu$ so that $Lc_Y(P_\mu(X-bY,Y)) \in \mathbb{Z}_\mu$
Triangular decomposition: $\{(A_i(X), B_i(X,Y))\}_{i\in\mathcal{I}}$ = Algorithm 2 $(P_\mu, Q_\mu, 0)$
for all $i \in \mathcal{I}$ do
    $C_i(X) = Lc_Y(B_i(X,Y))^{-1} \bmod A_i(X)$
    $\tilde{B}_i(X,Y) = C_i(X)B_i(X,Y) \bmod A_i(X)$
    Triangular decomp.: $\{(A_{ij}(X), B_{ij}(X,Y))\}_{j\in\mathcal{J}_i}$ = Algorithm 2 $\left(\tilde{B}_i(X,Y), \frac{\partial \tilde{B}_i(X,Y)}{\partial Y}, A_i(X)\right)$
end for
return $\sum_{i\in\mathcal{I}} \left(i\, d_X(A_i) - \sum_{j\in\mathcal{J}_i} j\, d_X(A_{ij})\right)$
```

of $\langle P_\mu, Q_\mu\rangle$ is the disjoint union of the solutions of the $\langle A_i(X), B_i(X,Y)\rangle$, for $i \in \mathcal{I}$. It follows that the number of (distinct) solutions of $I_\mu = \langle P_\mu, Q_\mu\rangle$ is

$$\#V(I_\mu) = \sum_{i\in\mathcal{I}} \sum_{\alpha\in V(A_i)} d_Y(\overline{B_i(\alpha, Y)}).$$

Since $B_i(\alpha, Y)$ is a univariate polynomial in Y, $d_Y(\overline{B_i(\alpha,Y)}) = d_Y(B_i(\alpha,Y)) - d_Y(\gcd(B_i(\alpha,Y), B_i'(\alpha,Y)))$, where $B_i'(\alpha,Y)$ is the derivative of $B_i(\alpha,Y)$, which is also equal to $\frac{\partial B_i}{\partial Y}(\alpha,Y)$. By Lemma 3.5.1, $d_Y(B_i(\alpha,Y)) = i$, and since the degree of the gcd is zero when $B_i(\alpha,Y)$ is squarefree, we have

$$\#V(I_\mu) = \sum_{i\in\mathcal{I}} \left(\sum_{\alpha\in V(A_i)} i - \sum_{\substack{\alpha\in V(A_i)\\ B_i(\alpha,Y)\text{ not sqfr.}}} d_Y(\gcd(B_i(\alpha,Y), \tfrac{\partial B_i}{\partial Y}(\alpha,Y))) \right). \tag{3.5}$$

The polynomials $A_i(X)$ are squarefree by Lemma 3.5.1, so $\sum_{\alpha\in V(A_i)} i$ is equal to $i\, d_X(A_i)$.

We now consider the sum of the degrees of the gcds. The rough idea is to apply Algorithm 2 to $B_i(X,Y)$ and $\frac{\partial B_i}{\partial Y}(X,Y)$, for every $i \in \mathcal{I}$, which computes a triangular decomposition $\{(A_{ij}(X), B_{ij}(X,Y))\}_{j\in\mathcal{J}_i}$ such that, for $\alpha \in V(A_{ij})$, $d_Y(\gcd(B_i(\alpha,Y), \frac{\partial B_i}{\partial Y}(\alpha,Y))) = j$ (by Lemma 3.5.1), which simplifies Equation (3.5) into $\#V(I_\mu) = \sum_{i\in\mathcal{I}} \left(i\, d_X(A_i) - \sum_{j\in\mathcal{J}_i} \sum_{\alpha\in V(A_{ij})} j\right)$. However, we cannot directly apply Algorithm 2 to $B_i(X,Y)$ and $\frac{\partial B_i}{\partial Y}(X,Y)$ because their leading coefficients in Y have no reasons to be coprime.

By Lemma 3.5.1, $A_i(X)$ and $Lc_Y(B_i(X,Y))$ are coprime, thus $Lc_Y(B_i(X,Y))$ is invertible modulo $A_i(X)$ (by BÃ©zout's identity); let $C_i(X)$ be this inverse and define $\tilde{B}_i(X,Y) = C_i(X)B_i(X,Y) \bmod A_i(X)$ (such that every coefficient of $C_i(X)B_i(X,Y)$ with respect to Y is reduced modulo $A_i(X)$). The leading coefficient in Y of $\tilde{B}_i(X,Y)$ is equal to 1, so we can apply Algorithm 2 to $\tilde{B}_i(X,Y)$ and $\frac{\partial \tilde{B}_i}{\partial Y}(X,Y)$. Furthermore, if $A_i(\alpha) = 0$, then $\tilde{B}_i(\alpha,Y) = C_i(\alpha)B_i(\alpha,Y)$ where $C_i(\alpha) \neq 0$ since $C_i(\alpha)Lc_Y(B_i(\alpha,Y)) = 1$. Equation (3.5) can thus be rewritten by replacing B_i by $\tilde{B}_i$.

By Lemma 3.5.1, for every $i \in \mathcal{I}$, Algorithm 2 computes a triangular decomposition $\{(A_{ij}(X),\ B_{ij}(X,Y))\}_{j\in\mathcal{J}_i}$ such that $V(\langle \tilde{B}_i, \frac{\partial \tilde{B}_i}{\partial Y}, A_i\rangle)$ is the disjoint union of the sets $V(\langle A_{ij}(X), B_{ij}(X,Y)\rangle)$, $j \in \mathcal{J}_i$, and for all $\alpha \in V(A_{ij})$, $d_Y(\gcd(\tilde{B}_i(\alpha,Y), \frac{\partial \tilde{B}_i}{\partial Y}(\alpha,Y))) = j$. Since the set of $\alpha \in V(A_i)$ such that $\tilde{B}_i(\alpha,Y)$ is not squarefree is the projection of the set of solutions $(\alpha,\beta) \in V(\langle \tilde{B}_i, \frac{\partial \tilde{B}_i}{\partial Y}, A_i\rangle)$ we get

$$\#V(I_\mu) = \sum_{i\in\mathcal{I}} \left(i\, d_X(A_i) - \sum_{j\in\mathcal{J}_i} \sum_{\alpha\in V(A_{ij})} j \right).$$

$A_{ij}(X)$ is squarefree (Lemma 3.5.1) so $\sum_{\alpha\in V(A_{ij})} j = j\, d_X(A_{ij})$, which concludes the proof. □

The next lemma gives the arithmetic complexity of the above algorithm.

Lemma 3.5.4. *Given P_μ, Q_μ in $\mathbb{Z}_\mu[X,Y]$ of total degree at most d, Algorithm 3 performs $\widetilde{O}(d^4)$ operations in $\mathbb{Z}_\mu$.*

Proof. According to Lemma 3.2.1, the sheared polynomials $P(T-SY,Y)$ and $Q(T-SY,Y)$ can be expanded in $\widetilde{O}_B(d^4+d^3\tau)$ bit operations in $\mathbb{Z}$. Thus the sheared polynomials $P_\mu(X-SY,Y)$ and $Q_\mu(X-SY,Y)$ can obviously be computed in $\widetilde{O}(d^4)$ arithmetic operations in $\mathbb{Z}_\mu$.[5] The leading term $Lc_Y(P_\mu(X-SY,Y)) \in \mathbb{Z}_\mu[S]$ is a polynomial of degree at most d and a value $b \in \mathbb{Z}_\mu$ that does not vanish it can be found by at most $d+1$ evaluations. Each evaluation can be done with $O(d)$ arithmetic operations, thus the shear value b can be computed in $\widetilde{O}(d^2)$ operations. It remains to evaluate the generically sheared polynomials at this value $S = b$. These polynomials have $O(d^2)$ monomials in X and Y, each with a coefficient in $\mathbb{Z}_\mu[S]$ of degree at most d; since the evaluation of each coefficient is softly linear in d, this gives a total complexity in $\widetilde{O}(d^4)$ for Line 1.

According to Lemma 3.5.2, the triangular decomposition in Line 2 can be done in $\widetilde{O}(d^4)$ arithmetic operations. In Lines 4 and 5, $C_i(X)$ and $\tilde{B}_i(X,Y)$

---

[5] It can easily be proved that these polynomials can be computed in $\widetilde{O}(d^3)$ arithmetic operations but the $\widetilde{O}(d^4)$ bound is sufficient here.

can be computed by first reducing modulo $A_i(X)$ every coefficient of $B_i(X,Y)$ (with respect to Y). There are at most i coefficients (by definition of subresultants) and according to Theorem 2.3.2, the arithmetic complexity of every reduction is softly linear in the degree of the operands, which is $\widetilde{O}(d^2)$ by Theorem 2.4.16. The reduction of $B_i(X,Y)$ modulo $A_i(X)$ can thus be done with $\widetilde{O}(d^3)$ arithmetic operations in $\mathbb{Z}_\mu$. Now, in Line 4, the arithmetic complexity of computing the inverse of one of these coefficients modulo $A_i(X)$ is softly linear in its degree [von zur Gathen 2003, Corollary 11.8], that is $\widetilde{O}(d_i)$ where d_i denotes the degree of $A_i(X)$. Furthermore, computing the product modulo $A_i(X)$ of two polynomials which are already reduced modulo $A_i(X)$ can be done in $\widetilde{O}(d_i)$ arithmetic operations [von zur Gathen 2003, Corollary 11.8]. Thus, in Line 5, the computation of $\widetilde{B}_i(X,Y)$ can be done with i such multiplications, and thus with $\widetilde{O}(id_i)$ arithmetic operations. Finally, in Line 6, the triangular decomposition can be done with $\widetilde{O}(i^3d_i)$ arithmetic operations by Lemma 3.5.2. The complexity of Lines 4-6 is thus in $\widetilde{O}(d^3 + i^3d_i)$ which is in $\widetilde{O}(d^3 + d^2id_i)$. The total complexity of the loop in Line 3 is thus $\widetilde{O}(d^4 + d^2\sum_i id_i)$ which is in $\widetilde{O}(d^4)$ because the number of solutions of the triangular system $(A_i(X), B_i(X,Y))$ is at most the degree of A_i times the degree of B_i in Y, that is id_i, and the total number of these solutions for $i \in \mathcal{I}$ is that of (P,Q), by Lemma 3.5.1, which is at most d^2 by BÃ©zout's bound. This concludes the proof because the sum in Line 8 can obviously be done in linear time in the size of the triangular decompositions that are computed during the algorithm. □

3.6 Computing a lucky prime and the number of solutions of I

We now show how to compute the number of solutions of $I = \langle P, Q\rangle$ and a lucky prime for that ideal.

Lemma 3.6.1. *Algorithm 4 computes the number of distinct solutions and a lucky prime for* $\langle P, Q\rangle$ *in* $\widetilde{O}_B(d^8 + d^7\tau)$ *bit operations. Moreover, this lucky prime is upper bounded by* $\widetilde{O}(d^4 + d^3\tau)$.

Proof. We first prove the correctness of the algorithm. Note first that for all $\mu \in B$ satisfying the constraint of Line 4, Lemma 3.3.4 implies that $\phi_\mu(P)$ and $\phi_\mu(Q)$ are coprime. It follows that Algorithm 3 computes the number of distinct solutions $N_\mu = \#V(I_\mu)$ of I_μ. By Proposition 3.4.1 and Definition 3.3.1, $N_\mu \leqslant \#V(I)$ and the equality holds if μ is lucky for I. Since the set B of considered primes contains a lucky one by construction, the maximum of the computed value of N_μ is equal to $\#V(I)$. Finally, the μ associated to any

Algorithm 4 Number of distinct solutions and lucky prime for $\langle P, Q\rangle$

```
Require: $P, Q$ in $\mathbb{Z}[X, Y]$ coprime of total degree at most $d$ and bitsize at most $\tau$
Ensure: The number of solutions and a lucky prime $\mu$ for $\langle P, Q\rangle$
1: Compute $P(T-SY, Y)$, $Q(T-SY, Y)$, $R(T, S) = Res_Y(P(T-SY, Y), Q(T-SY, Y))$
2: Compute a set $B$ of primes larger than $2d^4$ and of cardinality $\widetilde{O}(d^4+d^3\tau)$ that contains a lucky prime for $\langle P, Q\rangle$ (see Proposition 3.4.2)
3: for all $\mu$ in $B$ do
4:     if $\phi_\mu(L_P(S))\ \phi_\mu(L_Q(S))\ \phi_\mu(L_R(S)) \not\equiv 0$ then
5:         Compute $N_\mu$ = Algorithm 3($\phi_\mu(P), \phi_\mu(Q)$)
6:     end if
7: end for
8: return $(\mu, N_\mu)$ such that $N_\mu$ is maximum
```

such maximum value of N_μ is necessarily lucky by the constraint of Line 4 and since μ is larger than $2d^4$.

We now prove the complexity of the algorithm. The polynomials $P(T-SY, Y), Q(T-SY, Y)$ and their resultant $R(T, S)$ can be computed in $\widetilde{O}_B(d^7+d^6\tau)$ bit operations by Lemma 3.2.1.

Proposition 3.4.2 states that we can compute an explicit bound $\Xi(d, \tau)$ in $\widetilde{O}(d^4+d^3\tau)$ on the number of unlucky primes for $\langle P, Q\rangle$. We want to compute in Line 2 a set B of at least $\Xi(d, \tau)$ primes (plus one) that are larger than $2d^4$. For computing B, we can thus compute the first $\Xi(d, \tau) + 2d^4 + 1$ prime numbers and reject those that are smaller than $2d^4$. The bit complexity of computing the r first prime numbers is in $\widetilde{O}(r)$ and their maximum is in $\widetilde{O}(r)$ [von zur Gathen 2003, Theorem 18.10]. We can thus compute the set of primes B with $\widetilde{O}_B(d^4+d^3\tau)$ bit operations and these primes are in $\widetilde{O}(d^4+d^3\tau)$.

In Line 4, we test to zero the reduction modulo μ of three polynomials in $\mathbb{Z}[S]$ which have been computed in Line 1 and which are of degree $O(d^2)$ and bitsize $O(d^2+d\tau)$ in the worst case (by Lemma 3.2.1). For each of these polynomials, the test to zero can be done by first computing (once for all) the gcd of its $O(d^2)$ integer coefficients of bitsize $O(d^2+d\tau)$. Each gcd can be computed with a bit complexity that is softly linear in the bitsize of the integers [Yap 2000, §2.A.6] (and the bitsize clearly does not increase), hence all the gcds can be done with a bit complexity of $\widetilde{O}_B(d^2(d^2+d\tau))$. Then the reduction of each of the three gcds modulo all the primes in B can be computed via a remainder tree in a bit complexity that is softly linear in the total bitsize of the input (Theorem 2.5.3), which is dominated by the sum of the bitsizes of the $\widetilde{O}(d^4+d^3\tau)$ primes in B each of bitsize in $\widetilde{O}(1)$. Hence, the

```
**Algorithm 5** Separating form for $\langle P, Q\rangle$
Require: $P, Q$ in $\mathbb{Z}[X, Y]$ of total degree at most $d$ and defining a zero-dimensional ideal $I$
Ensure: A linear form $X + aY$ that separates $V(I)$, with $a < 2d^4$ and $L_P(a)\, L_Q(a) \neq 0$
 1: Apply Algorithm 4 to compute the number of solutions $\#V(I)$ and a lucky prime $\mu$ for $I$
 2: Compute $P(T - SY, Y)$, $Q(T - SY, Y)$ and $R(T, S) = Res_Y(P(T - SY, Y), Q(T - SY, Y))$
 3: Compute $R_\mu(T, S) = \phi_\mu(R(T, S))$
 4: Compute $\Upsilon_\mu(S) = \phi_\mu(L_P(S))\, \phi_\mu(L_Q(S))\, \phi_\mu(L_R(S))$
 5: $a := 0$
 6: repeat
 7:     Compute the degree $N_a$ of the squarefree part of $R_\mu(T, a)$
 8:     $a := a + 1$
 9: until $\Upsilon_\mu(a) \neq 0$[6]and $N_a = \#V(I)$
10: return The linear form $X + aY$
```

tests in Line 4 can be done with a total bit complexity in $\widetilde{O}_B(d^4 + d^3\tau)$.

In Line 5, we compute, for $\widetilde{O}(d^4 + d^3\tau)$ prime numbers μ, $\phi_\mu(P)$ and $\phi_\mu(Q)$ and call Algorithm 3 to compute the number of their common solutions. Using the same argument as above, each coefficient of P and Q can be reduced modulo the $\widetilde{O}(d^4 + d^3\tau)$ primes with $\widetilde{O}_B(d^4 + d^3\tau)$ bit operations, thus the bit complexity of computing all the $\phi_\mu(P)$ and $\phi_\mu(Q)$ for μ in B is in $\widetilde{O}_B(d^6 + d^5\tau)$. By Lemma 3.5.4, the bit complexity of Algorithm 3 is in $\widetilde{O}_B(d^4)$. Hence, the total bit complexity of Line 5 is $\widetilde{O}_B(d^8 + d^7\tau)$, and so is the overall bit complexity of Algorithm 4. $\square$

3.7 Computing a separating linear form

Using Algorithm 4, we now present our algorithm for computing a linear form that separates the solutions of $\langle P, Q\rangle$.

Theorem 3.7.1. *Algorithm 5 returns a separating linear form $X + aY$ for $\langle P, Q\rangle$ with $a < 2d^4$. The bit complexity of the algorithm is in $\widetilde{O}_B(d^8 + d^7\tau)$.*

Proof. We first prove the correctness of the algorithm. We start by proving that the value a returned by the algorithm is the smallest nonnegative integer such that $X + aY$ separates $V(I_\mu)$ with $\Upsilon_\mu(a) \neq 0$. Note first that, in Line 3,

---

[6] $\Upsilon_\mu(S)$ is a polynomial in $\mathbb{Z}_\mu[S]$ and we consider $\Upsilon_\mu(a)$ in $\mathbb{Z}_\mu$.

$\phi_\mu(R(T,S))$ is indeed equal to $R_\mu(T,S)$ which is defined as $Res_Y(P_\mu(T - SY,Y), Q_\mu(T - SY,Y))$ since the leading coefficients $L_P(S)$ and $L_Q(S)$ of $P(T - SY,Y)$ and $Q(T - SY,Y)$ do not identically vanish modulo μ (since μ is lucky), and thus $L_{P_\mu}(S) = \phi_\mu(L_P(S))$, similarly for Q, and according to Theorem 2.4.4 the resultant can be specialized modulo μ. Now, Line 9 ensures that the value a returned by the algorithm satisfies $\Upsilon_\mu(a) \neq 0$, and we restrict our attention to nonnegative such values of a. Note that $\Upsilon_\mu(a) \neq 0$ implies that $\phi_\mu(L_P(a))\ \phi_\mu(L_Q(a))\ \phi_\mu(L_R(a)) \neq 0$ because the specialization at $S = a$ and the reduction modulo μ commute (in $\mathbb{Z}_\mu$). For the same reason, $L_{P_\mu}(S) = \phi_\mu(L_P(S))$ implies $L_{P_\mu}(a) = \phi_\mu(L_P(a))$ and thus $L_{P_\mu}(a) \neq 0$ and, similarly, $L_{Q_\mu}(a) \neq 0$. On the other hand, Line 9 implies that the value a is the smallest that satisfies $d_T(\overline{R_\mu(T,a)}) = \#V(I)$, which is also equal to $\#V(I_\mu)$ since μ is lucky. Lemma 3.3.3 thus yields that the returned value a is the smallest nonnegative integer such that $X + aY$ separates $V(I_\mu)$ and $\Upsilon_\mu(a) \neq 0$, which is our claim.

This property first implies that $a < 2d^4$ because the degree of Υ_μ is bounded by $2(d^2+d)$, the number of non-separating linear forms is bounded by $\binom{d^2}{2}$ (the maximum number of directions defined by any two of d^2 solutions), and their sum is less than $2d^4$ for $d \geqslant 2$. Note that, since μ is lucky, $2d^4 < \mu$ and thus $a < \mu$. The above property thus also implies, by Proposition 3.3.2, that $X + aY$ separates $V(I)$. This concludes the proof of correctness of the algorithm since $a < 2d^4$ and $L_P(a)\, L_Q(a) \neq 0$ (since $\Upsilon_\mu(a) \neq 0$).

We now focus on the complexity of the algorithm. By Lemma 3.6.1, the bit complexity of Line 1 is in $\widetilde{O}_B(d^8 + d^7\tau)$. The bit complexity of Lines 2 to 5 is in $\widetilde{O}_B(d^7 + d^6\tau)$. Indeed, by Lemma 3.2.1, $R(T,S)$ has degree $O(d^2)$ in T and in S, bitsize $\widetilde{O}(d^2 + d\tau)$, and it can be computed in $\widetilde{O}_B(d^7 + d^6\tau)$ time. Computing $R_\mu(T,S) = \phi_\mu(R(T,S))$ can thus be done in reducing $O(d^4)$ integers of bitsize $\widetilde{O}(d^2 + d\tau)$ modulo μ. According to Theorem 2.3.2, each reduction is softly linear in the maximum of the bitsizes, thus the reduction of $R(T,S)$ can be computed in $\widetilde{O}_B(d^4(d^2 + d\tau))$ time (since μ has bitsize in $O(\log(d^4 + d^3\tau))$ by Lemma 3.6.1).[7] The computation of Υ_μ can clearly be done with the same complexity since each reduction is easier than the one in Line 3, and the product of the polynomials (which does not actually need to be computed since we are only interested in whether $\Upsilon_\mu(a)$ vanishes) can be done with a bit complexity that is softly linear in the product of the maximum degrees and maximum bitsizes according to Theorem 2.3.1.

---

[7] Note that $R_\mu(T,S)$ can be computed more efficiently in $\widetilde{O}_B(d^5 + d^3\tau)$ bit operations as the resultant of $P_\mu(T-SY,Y)$ and $Q_\mu(T-SY,Y)$ because computing these two polynomials and their reduction can be done in $\widetilde{O}_B(d^4 + d^3\tau)$ bit operations (Lemma 3.2.1) and their resultant can be computed with $\widetilde{O}(d^5)$ arithmetic operations in $\mathbb{Z}_\mu$ (Lemma 2.4.17) and thus with $\widetilde{O}_B(d^5)$ bit operations since μ has bitsize in $O(\log(d^4 + d^3\tau))$.

We proved that the value a returned by the algorithm is less than $2d^4$, thus the loop in Line 6 is performed at most $2d^4$ times. Each iteration consists of computing the squarefree part of $R_\mu(T,a)$ which requires $\widetilde{O}_B(d^4)$ bit operations. Indeed, computing $R_\mu(T,S)$ at $S=a$ amounts to evaluating, in $\mathbb{Z}_\mu$, $O(d^2)$ polynomials in S, each of degree $O(d^2)$ (by Lemma 3.2.1). Note that a does not need to be reduced modulo μ because $a<2d^4$ and $2d^4<\mu$ since μ is lucky. Thus, the bit complexity of evaluating in $\mathbb{Z}_\mu$ each of the $O(d^2)$ polynomials in S is the number of arithmetic operations in $\mathbb{Z}_\mu$, which is linear the degree that is $O(d^2)$, times the (maximum) bit complexity of the operations in $\mathbb{Z}_\mu$, which is in $O_B(\log d\tau)$ since μ is in $\widetilde{O}(d^4+d^3\tau)$ by Lemma 3.6.1. Hence, computing $R_\mu(T,a)$ can be done in $\widetilde{O}_B(d^4)$ bit operations. Once $R_\mu(T,a)$ is computed, the arithmetic complexity of computing its squarefree part in $\mathbb{Z}_\mu$ is softly linear in its degree (Theorem 2.3.8), that is $\widetilde{O}(d^2)$, which yields a bit complexity in $\widetilde{O}_B(d^2)$ since, again, μ is in $\widetilde{O}(d^4+d^3\tau)$. This leads to a total bit complexity of $\widetilde{O}_B(d^8)$ for the loop in Lines 6 to 9, and thus to a total bit complexity for the algorithm in $\widetilde{O}_B(d^8+d^7\tau)$. □

3.8 Conclusion

We presented an algorithm of bit complexity $\widetilde{O}_B(d^8+d^7\tau)$ for finding a separating linear form of a bivariate system, improving by a factor d^2 the best known algorithm for this problem. Finding a separating linear form is at the core of approaches based on rational parametrizations for solving such systems and, as mentioned in the introduction, our algorithm directly improves the bit complexity of the classical method for computing rational parametrizations via subresultants [Gonzalez-Vega 1996]. Interestingly, computing a separating linear form remains the bit-complexity bottleneck in this algorithm [Diochnos 2009] and we will show in the next chapter that this is also the bottleneck for computing the rational parametrization of [Rouillier 1999]. Another result proved in Chapter 6, is that isolating boxes for the solutions of a bivariate system can be computed from these rational parametrizations in a smaller bit complexity. This thus yields algorithms of bit complexity $\widetilde{O}_B(d^8+d^7\tau)$ for computing rational parametrizations of bivariate systems and isolating their real solutions. It should be stressed that this complexity matches the recent one presented by Emeliyanenko and Sagraloff [Emeliyanenko 2012] for "only" computing isolating boxes of the real solutions. Furthermore, as shown in Chapter 6, rational parametrizations yield efficient algorithms for various related problems, such as evaluating the sign of a polynomial at the solutions of the system, or solving over-constrained systems.

CHAPTER 4

Rational Univariate Representation

Contents

In this chapter we present and analyse an algorithm for computing the Rational Univariate Representation (RUR) of a system defined by two bivariate polynomials of total degree at most d with integer coefficients of maximum bitsize τ. As mentioned in the introduction, such an algorithm decomposes in two phases: first computing a separating linear form for the solutions and second, computing the associated Rational Univariate Representation. In the previous chapter, we focused on the first phase of this algorithm and showed that a separating linear form can be computed in $\widetilde{O}_B(d^8+d^7\tau)$ bit operations. Here, we suppose computed a separating linear form and focus on the second phase, that is the computation of the Rational Univariate Representation.

The results presented in this chapter are part of an article published in the ISSAC 2013 conference [Bouzidi 2013a].

4.1 Introduction

We first show that the Rational Univariate Representation (RUR for short) of Rouillier [Rouillier 1999] (i) can be expressed with simple polynomial formulas, that (ii) it has a total bitsize which is asymptotically smaller than that of Gonzalez-Vega and El Kahoui [Gonzalez-Vega 1996] by a factor d, and that (iii) it can be computed with the same complexity, that is $\widetilde{O}_B(d^7 + d^6\tau)$ (Theorem 4.1.1). Namely, we prove that the RUR consists of four polynomials of degree at most d^2 and bitsize $\widetilde{O}(d^2 + d\tau)$ (instead of $O(d)$ polynomials with the same asymptotic degree and bitsize for Gonzalez-Vega and El Kahoui parametrization). Moreover, we prove that this bound holds for any ideal containing P and Q, that is, for instance the radical ideal of $\langle P, Q\rangle$ (Proposition 4.3.1).

We formally state in the following theorem the main result of this chapter. This theorem is stated for any separating linear form $X+aY$ with integer a of bitsize $\widetilde{O}(1)$ with the abuse of notation that polylogarithmic factors in d and τ are omitted. Recall that according to the previous chapter, a separating form $X+aY$ with a positive integer $a < 2d^4$ can be computed in $\widetilde{O}_B(d^8 + d^7\tau)$ bit operations (Theorem 3.7.1). This theorem is a direct consequence of Propositions 4.2.2 and 4.3.1.

Theorem 4.1.1. *Let $P, Q \in \mathbb{Z}[X, Y]$ be two coprime bivariate polynomials of total degree at most d and maximum bitsize τ. Given a separating form $X + aY$ with integer a of bitsize $\widetilde{O}(1)$, the RUR of $\langle P, Q\rangle$ associated to a can be computed using Proposition 4.2.1 with $\widetilde{O}_B(d^7 + d^6\tau)$ bit operations. Furthermore, the polynomials of this RUR have degree at most d^2 and bitsize in $\widetilde{O}(d^2 + d\tau)$.*

This chapter is organized as follows: in Section 4.2, we present our algorithm for computing the RUR based on the formulas of Proposition 4.2.1. We then use these formulas in Section 4.3 to prove new bounds on the bitsize of the coefficients of the polynomials of the RUR. The main results of this chapter are summarized in Theorem 4.1.1.

Throughout this chapter we assume that the two input polynomials P and Q are coprime in $\mathbb{Z}[X, Y]$, that their maximum total degree d is at least 2 and that their coefficients have maximum bitsize τ.

4.2 RUR computation

We show here that the polynomials of a RUR can be expressed as combinations of specializations of the resultant R and its partial derivatives. The seminal idea has already been used by several authors in various contexts (see e.g. [Canny 1987, Alonso 1996, Schost 2001]) for computing rational parameterizations of the radical of a given zero-dimensional ideal and mainly for bounding the size of a Chow form. Based on the same idea but keeping track of multiplicities, we present a simple new formulation for the polynomials of a RUR, given a separating form.

For the convenience of the reader, we recall here the definition of the Rational Univariate Representation of a bivariate system we introduced in Section 2.2.

Definition 2.2.1. *Let $I \subset \mathbb{Q}[X,Y]$ be a zero-dimensional ideal, $V(I) = \{\sigma \in \mathbb{C}^2, v(\sigma) = 0, \forall v \in I\}$ its associated variety, and a linear form $T = X + aY$ with $a \in \mathbb{Q}$. The RUR-candidate of I associated to $X + aY$ (or simply, to a), denoted $RUR_{I,a}$, is the following set of four univariate polynomials in $\mathbb{Q}[T]$*

$$\begin{aligned} f_{I,a}(T) &= \prod_{\sigma \in V(I)} (T - X(\sigma) - aY(\sigma))^{\mu_I(\sigma)} \\ f_{I,a,v}(T) &= \sum_{\sigma \in V(I)} \mu_I(\sigma) v(\sigma) \prod_{\varsigma \in V(I), \varsigma \neq \sigma} (T - X(\varsigma) - aY(\varsigma)), \quad \text{for } v \in \{1, X, Y\} \end{aligned} \tag{4.1}$$

where, for $\sigma \in V(I)$, $\mu_I(\sigma)$ denotes the multiplicity of σ in I. If $(X,Y) \mapsto X + aY$ is injective on $V(I)$, we say that the linear form $X + aY$ separates $V(I)$ (or is separating for I) and $RUR_{I,a}$ is called a RUR (the RUR of I associated to a) and it defines a bijection between $V(I)$ and $V(f_{I,a}) = \{\gamma \in \mathbb{C}, f_{I,a}(\gamma) = 0\}$:

$$\begin{array}{ccc} V(I) & \to & V(f_{I,a}) \\ (\alpha, \beta) & \mapsto & \alpha + a\beta \\ \left(\dfrac{f_{I,a,X}}{f_{I,a,1}}(\gamma), \dfrac{f_{I,a,Y}}{f_{I,a,1}}(\gamma)\right) & \hookleftarrow & \gamma \end{array}$$

Moreover, this bijection preserves the real roots and the multiplicities.

Proposition 4.2.1. *For any rational a such that $L_P(a)L_Q(a) \neq 0$ and such that $X + aY$ is a separating form of $I = \langle P, Q \rangle$, the RUR of $\langle P, Q \rangle$ associated to a is as follows:*

$$\begin{aligned} f_{I,a}(T) &= \frac{R(T,a)}{L_R(a)} \\ f_{I,a,1}(T) &= \frac{f'_{I,a}(T)}{\gcd(f_{I,a}(T), {f'}_{I,a}(T))} \\ f_{I,a,Y}(T) &= \frac{\frac{\partial R}{\partial S}(T,a) - f_{I,a}(T) \frac{\partial L_R}{\partial S}(a)}{L_R(a) \gcd(f_{I,a}(T), {f'}_{I,a}(T))} \\ f_{I,a,X}(T) &= T f_{I,a,1}(T) - d_T(f_{I,a}) \overline{f_{I,a}(T)} - a f_{I,a,Y}(T). \end{aligned}$$

We postpone the proof of Proposition 4.2.1 to Section 4.2.1 and first analyze the complexity of the computation of the expressions therein. Note that a separating form $X + aY$ as in Proposition 4.2.1 can be computed in $\widetilde{O}_B(d^8 + d^7\tau)$ according to Theorem 3.7.1.

Proposition 4.2.2. *Computing the polynomials in Proposition 4.2.1 can be done with $\widetilde{O}_B(d^7 + d^6(\tau + \tau_a))$ bit operations, where τ_a is the bitsize of a.*

Proof of Proposition 4.2.2. According to Lemma 3.2.1, the resultant $R(T,S)$ of $P(T-SY,Y)$ and $Q(T-SY,Y)$ with respect to Y has degree $O(d^2)$ in T and S, has bitsize in $\widetilde{O}(d(d+\tau))$, and that it can be computed in $\widetilde{O}_B(d^6(d+\tau))$ bit operations. We can now apply the formulas of Proposition 4.2.1 for computing the polynomials of the RUR.

Specializing $R(T,S)$ at $S=a$ can be done by evaluating $O(d^2)$ polynomials in S, each of degree in $O(d^2)$ and bitsize in $\widetilde{O}(d^2+d\tau)$. By Theorem 2.3.4, each of the $O(d^2)$ evaluations can be done in $\widetilde{O}_B(d^2(d^2+d\tau+\tau_a))$ bit operations and each result has bitsize in $\widetilde{O}(d^2+d\tau+d^2\tau_a)$. Hence, $R(T,a)$ and $f_{I,a}(T)$ have degree in $O(d^2)$, bitsize in $\widetilde{O}(d^2+d\tau+d^2\tau_a)$, and they can be computed with $\widetilde{O}_B(d^4(d^2+d\tau+\tau_a))$ bit operations.

The complexity of computing the numerators of $f_{I,a,1}(T)$ and $f_{I,a,Y}(T)$ is clearly dominated by the computation of $\frac{\partial R}{\partial S}(T,a)$. Indeed, computing the derivative $\frac{\partial R}{\partial S}(T,S)$ can trivially be done in $O(d^4)$ arithmetic operations of complexity $\widetilde{O}_B(d^2+d\tau)$, that is in $\widetilde{O}_B(d^6+d^5\tau)$. Then, as for $R(T,a)$, $\frac{\partial R}{\partial S}(T,a)$ has degree in $O(d^2)$, bitsize in $\widetilde{O}(d^2+d\tau+d^2\tau_a)$, and it can be computed within the same complexity as the computation of $R(T,a)$.

On the other hand, since $f_{I,a}(T)$ and $f'_{I,a}(T)$ have degree in $O(d^2)$ and bitsize in $\widetilde{O}(d^2+d\tau+d^2\tau_a)$, and $f_{I,a}(T)=\frac{R(T,a)}{L_R(a)}$, one can multiply these two polynomials by $L_R(a)$ which is of bitsize $\widetilde{O}(d^2+d\tau+d^2\tau_a)$ and by the denominator of the rational a to the power of $d_S(R(T,S))$ which is an integer of bitsize in $O(d^2\tau_a)$, to obtain polynomials with coefficients in $\mathbb{Z}$. Hence, according to Theorem 2.3.10, the gcd of $f_{I,a}(T)$ and $f'_{I,a}(T)$ can be computed in $\widetilde{O}_B(d^4(d^2+d\tau+d^2\tau_a))$ bit operations and it has bitsize in $\widetilde{O}(d^2+d\tau+d^2\tau_a)$.

Now, according to Theorem 2.3.2 the bit complexity of the division of the numerators by the gcd is of the order of the square of their maximum degree times their maximum bitsize which is in $\widetilde{O}_B(d^4(d^2+d\tau+d^2\tau_a))$ bit operations.

Finally, computing $f_{I,a,X}(T)$ can be done within the same complexity as for $f_{I,a,1}(T)$ and $f_{I,a,Y}(T)$ since it is dominated by the computation of the squarefree part of $f_{I,a}(T)$, which can be computed similarly and with the same complexity as above, by Theorem 2.3.10.

The overall complexity is thus that of computing the resultant which is in $\widetilde{O}_B(d^6(d+\tau))$ plus that of computing the above gcd and Euclidean division which is in $\widetilde{O}_B(d^4(d^2+d\tau+d^2\tau_a))$. This gives a total of $\widetilde{O}_B(d^7+d^6(\tau+\tau_a))$. □

4.2.1 Proof of Proposition 4.2.1

Proposition 4.2.1 expresses the polynomials $f_{I,a}$ and $f_{I,a,v}$ of a RUR in terms of specializations (by $S=a$) of the resultant $R(T,S)$ and its partial derivatives. Since the specializations are done after considering the derivatives of R, we study the relations between these entities before specializing S by a.

For that purpose, we first introduce the following polynomials which are exactly the polynomials $f_{I,a}$ and $f_{I,a,v}$ of (4.1) where the parameter a is replaced by the variable S. These polynomials can be seen as the RUR polynomials of the ideal I with respect to a "generic" linear form $X + SY$.

$$\begin{aligned} f_I(T,S) &= \prod_{\sigma \in V(I)} (T - X(\sigma) - SY(\sigma))^{\mu_I(\sigma)} \\ f_{I,v}(T,S) &= \sum_{\sigma \in V(I)} \mu_I(\sigma) v(\sigma) \prod_{\varsigma \in V(I), \varsigma \neq \sigma} (T - X(\varsigma) - SY(\varsigma)), \quad v \in \{1, X, Y\}. \end{aligned} \tag{4.2}$$

These polynomials are obviously in $\mathbb{C}[T,S]$, but they are actually in $\mathbb{Q}[T,S]$ because, when S is specialized at any rational value a, the specialized polynomials are those of $RUR_{I,a}$ which are known to be in $\mathbb{Q}[T]$ (see e.g. [Rouillier 1999]).

Before proving Proposition 4.2.1, we express the derivatives of $f_I(T,S)$ in terms of $f_{I,v}(T,S)$, in Lemma 4.2.3, and show that $f_I(T,S)$ is the monic form of the resultant $R(T,S)$, seen as a polynomial in T, in Lemma 4.2.5.

Lemma 4.2.3. *Let* $g_I(T,S) = \prod_{\sigma \in V(I)} (T - X(\sigma) - SY(\sigma))^{\mu_I(\sigma)-1}$. *We have*

$$\frac{\partial f_I}{\partial T}(T,S) = g_I(T,S) f_{I,1}(T,S), \tag{4.3}$$

$$\frac{\partial f_I}{\partial S}(T,S) = g_I(T,S) f_{I,Y}(T,S). \tag{4.4}$$

Proof. It is straightforward that the derivative of f_I with respect to T is $\sum_{\sigma \in V(I)} \mu_I(\sigma)(T - X(\sigma) - SY(\sigma))^{\mu_I(\sigma)-1} \prod_{\varsigma \in V(I), \varsigma \neq \sigma} (T - X(\varsigma) - SY(\sigma))^{\mu_I(\varsigma)}$, which can be rewritten as the product of $\prod_{\sigma \in V(I)} (T - X(\sigma) - SY(\sigma))^{\mu_I(\sigma)-1}$ and $\sum_{\sigma \in V(I)} \mu_I(\sigma) \prod_{\varsigma \in V(I), \varsigma \neq \sigma} (T - X(\varsigma) - SY(\varsigma))$ which is exactly the product of $g_I(T,S)$ and $f_{I,1}(T,S)$.

The expression of the derivative of f_I with respect to S is similar to that with respect to T except that the derivative of $T - X(\sigma) - SY(\sigma)$ is now $Y(\sigma)$ instead of 1. It follows that $\frac{\partial f_I}{\partial S}$ is the product of $\prod_{\sigma \in V(I)} (T - X(\sigma) - SY(\sigma))^{\mu_I(\sigma)-1}$ and $\sum_{\sigma \in V(I)} \mu_I(\sigma) Y(\sigma) \prod_{\varsigma \in V(I), \varsigma \neq \sigma} (T - X(\varsigma) - SY(\varsigma))$ which is the product of $g_I(T,S)$ and $f_{I,Y}(T,S)$. □

For the proof of Lemma 4.2.5, we will need the following lemma which states that when two polynomials have no common solution at infinity in some direction, the roots of their resultant with respect to this direction are the projections of the solutions of the system with cumulated multiplicities.

Lemma 4.2.4 ([Busé 2005, Prop. 2 and 5])**.** *Let $P, Q \in \mathbb{F}[X,Y]$ defining a zero-dimensional ideal $I = \langle P, Q\rangle$, such that their leading terms $Lc_Y(P)$ and $Lc_Y(Q)$ do not have common roots. Then $Res_Y(P,Q) = c\prod_{\sigma\in V(I)}(X - X(\sigma))^{\mu_I(\sigma)}$ where c is nonzero in $\mathbb{F}$.*

The following lemma links the resultant of $P(T-SY,Y)$ and $Q(T-SY,Y)$ with respect to Y and the polynomial $f_I(T,S)$ as defined above.

Lemma 4.2.5. *$R(T,S) = L_R(S)f_I(T,S)$ and, for any $a \in \mathbb{Q}$, $L_P(a)L_Q(a) \neq 0$ implies that $L_R(a) \neq 0$.*

Proof. The proof is organized as follows. We first prove that for any rational a such that $L_P(a)L_Q(a)$ does not vanish, $R(T,a) = c(a)f_I(T,a)$ where $c(a) \in \mathbb{Q}$ is a nonzero constant depending on a. This is true for infinitely many values of a and, since $R(T,S)$ and $f_I(T,S)$ are polynomials, we can deduce that $R(T,S) = L_R(S)f_I(T,S)$. This will also implies the second statement of the lemma since, if $L_P(a)L_Q(a) \neq 0$, then $R(T,a) = c(a)f_I(T,a) = L_R(a)f_I(T,a)$ with $c(a) \neq 0$, thus $L_R(a) \neq 0$ (since $f_I(T,a)$ is monic).

If a is such that $L_P(a)L_Q(a) \neq 0$, then according to Theorem 2.4.4 the resultant $R(T,S)$ can be specialized at $S = a$, in the sense that $R(T,a)$ is equal to the resultant of $P(T-aY,Y)$ and $Q(T-aY,Y)$ with respect to Y.

We now apply Lemma 4.2.4 to these two polynomials $P(T-aY,Y)$ and $Q(T-aY,Y)$. These two polynomials satisfy the hypotheses of this lemma: first, their leading coefficients (in Y) do not depend on T, hence they have no common root in $\mathbb{Q}[T]$; second, the polynomials $P(T-aY,Y)$ and $Q(T-aY,Y)$ are coprime because $P(X,Y)$ and $Q(X,Y)$ are coprime by assumption and the change of variables $(X,Y) \mapsto (T = X + aY, Y)$ is a one-to-one mapping (and a common factor will remain a common factor after the change of variables). Hence Lemma 4.2.4 yields that $R(T,a) = c(a)\prod_{\sigma\in V(I_a)}(T-T(\sigma))^{\mu_{I_a}(\sigma)}$, where $c(a) \in \mathbb{Q}$ is a nonzero constant depending on a, and I_a is the ideal generated by $P(T-aY,Y)$ and $Q(T-aY,Y)$.

We now observe that $\prod_{\sigma\in V(I_a)}(T-T(\sigma))^{\mu_{I_a}(\sigma)}$ is equal to $f_I(T,a) = \prod_{\sigma\in V(I)}(T - X(\sigma) - aY(\sigma))^{\mu_I(\sigma)}$ since any solution (α,β) of $P(X,Y)$ is in one-to-one correspondence with the solution $(\alpha + a\beta, \beta)$ of $P(T-aY,Y)$ (and similarly for Q) and the multiplicities of the solutions also match, i.e. $\mu_I(\sigma) = \mu_{I_a}(\sigma_a)$ when σ and σ_a are in correspondence through the mapping [Fulton 2008, §3.3 Proposition 3 and Theorem 3]. Hence,

$$L_P(a)L_Q(a) \neq 0 \quad \Rightarrow \quad R(T,a) = c(a)f_I(T,a) \quad \text{with} \quad c(a) \neq 0. \tag{4.5}$$

Since there is finitely many values of a such that $L_P(a)L_Q(a)L_R(a) = 0$ and since $f_I(T,S)$ is monic with respect to T, (4.5) implies that $R(T,S)$ and

$f_I(T,S)$ have the same degree in T, say D. We write these two polynomials as

$$R(T,S) = L_R(S)T^D + \sum_{i=0}^{D-1} r_i(S)T^i, \qquad f_I(T,S) = T^D + \sum_{i=0}^{D-1} f_i(S)T^i. \tag{4.6}$$

If a is such that $L_P(a)L_Q(a)L_R(a) \neq 0$, (4.5) and (4.6) imply that $L_R(a) = c(a)$ and $r_i(a) = L_R(a)f_i(a)$, for all i. These equalities hold for infinitely many values of a, and $r_i(S), L_R(S)$ and $f_i(S)$ are polynomials in S, thus $r_i(S) = L_R(S)f_i(S)$ and, by (4.6), $R(T,S) = L_R(S)f_I(T,S)$. ☐

We can now prove Proposition 4.2.1, which we recall, for clarity.

Proposition 4.2.1. *For any rational a such that $L_P(a)L_Q(a) \neq 0$ and such that $X + aY$ is a separating form of $I = \langle P, Q\rangle$, the RUR of $\langle P, Q\rangle$ associated to a is as follows:*

$$f_{I,a}(T) = \frac{R(T,a)}{L_R(a)}$$

$$f_{I,a,1}(T) = \frac{f'_{I,a}(T)}{\gcd(f_{I,a}(T), f'_{I,a}(T))}$$

$$f_{I,a,Y}(T) = \frac{\frac{\partial R}{\partial S}(T,a) - f_{I,a}(T)\frac{\partial L_R}{\partial S}(a)}{L_R(a)\gcd(f_{I,a}(T), f'_{I,a}(T))}$$

$$f_{I,a,X}(T) = Tf_{I,a,1}(T) - d_T(f_{I,a})\overline{f_{I,a}(T)} - af_{I,a,Y}(T).$$

Proof. Since we assume that a is such that $L_P(a)L_Q(a) \neq 0$, Lemma 4.2.5 immediately gives the first formula.

Equation 4.3 states that $f_{I,1}(T,S)g_I(T,S) = \frac{\partial f_I(T,S)}{\partial T}$, where $g_I(T,S) = \prod_{\sigma\in V(I)}(T - X(\sigma) - SY(\sigma))^{\mu_I(\sigma)-1}$. In addition, g_I being monic in T, it never identically vanishes when S is specialized, thus the preceding formula yields after specialization: $f_{I,a,1}(T) = \frac{f'_{I,a}(T)}{g_I(T,a)}$. Furthermore, $g_I(T,a) = \gcd(f_{I,a}(T), f'_{I,a}(T))$. Indeed, $f_{I,a}(T) = \prod_{\sigma\in V(I)}(T - X(\sigma) - aY(\sigma))^{\mu_I(\sigma)}$ and all values $X(\sigma) + aY(\sigma)$, for $\sigma \in V(I)$, are pairwise distinct since $X + aY$ is a separating form, thus the gcd of $f_{I,a}(T)$ and its derivative is $\prod_{\sigma\in V(I)}(T - X(\sigma) - aY(\sigma))^{\mu_I(\sigma)-1}$, that is $g_I(T,a)$. This proves the formula for $f_{I,a,1}$.

Concerning the third equation, Lemma 4.2.5 together with Equation 4.4 implies:

$$\begin{aligned} f_{I,Y}(T,S) &= \frac{\frac{\partial f_I(T,S)}{\partial S}}{g_I(T,S)} = \frac{\frac{\partial (R(T,S)/L_R(S))}{\partial S}}{g_I(T,S)} = \frac{\frac{\partial R(T,S)}{\partial S}L_R(S) - R(T,S)\frac{\partial L_R(S)}{\partial S}}{L_R(S)^2 g_I(T,S)} \\ &= \frac{\frac{\partial R(T,S)}{\partial S} - f_I(T,S)\frac{\partial L_R(S)}{\partial S}}{L_R(S)g_I(T,S)}. \end{aligned}$$

As argued above, when specialized, $g_I(T,a) = \gcd(f_{I,a}(T), f'_{I,a}(T))$ and it does not identically vanish. By Lemma 4.2.5, $L_R(a)$ does not vanish either, and the formula for $f_{I,a,Y}$ follows.

It remains to compute $f_{I,a,X}$. Definition 2.2.1 implies that, for any root γ of $f_{I,a}$: $\gamma = \frac{f_{I,a,X}}{f_{I,a,1}}(\gamma) + a\frac{f_{I,a,Y}}{f_{I,a,1}}(\gamma)$, and thus $f_{I,a,X}(\gamma) + af_{I,a,Y}(\gamma) - \gamma f_{I,a,1}(\gamma) = 0$. Replacing γ by T, we have that the polynomial $f_{I,a,X}(T) + af_{I,a,Y}(T) - Tf_{I,a,1}(T)$ vanishes at every root of $f_{I,a}$, thus the squarefree part of $f_{I,a}$ divides that polynomial. In other words, $f_{I,a,X}(T) = Tf_{I,a,1}(T) - af_{I,a,Y}(T) \bmod \overline{f_{I,a}(T)}$. We now compute $Tf_{I,a,1}(T)$ and $af_{I,a,Y}(T)$ modulo $\overline{f_{I,a}(T)}$.

Equation (4.1) implies that $f_{I,a,v}(T)$ is equal to $T^{\#V(I)-1}\sum_{\sigma\in V(I)}\mu_I(\sigma)v(\sigma)$ plus some terms of lower degree in T, and that the degree of $\overline{f_{I,a}(T)}$ is $\#V(I)$ (since $X + aY$ is a separating form). First, for $v = Y$, this implies that $d_T(f_{I,a,Y}) < d_T(\overline{f_{I,a}})$, and thus that $af_{I,a,Y}(T)$ is already reduced modulo $\overline{f_{I,a}(T)}$. Second, for $v = 1$, $\sum_{\sigma\in V(I)}\mu_I(\sigma)$ is nonzero and equal to $d_T(f_{I,a})$. Thus, $Tf_{I,a,1}(T)$ and $\overline{f_{I,a}(T)}$ are both of degree $\#V(I)$, and their leading coefficients are $d_T(f_{I,a})$ and 1, respectively. Hence $Tf_{I,a,1}(T) \bmod \overline{f_{I,a}(T)} = Tf_{I,a,1}(T) - d_T(f_{I,a})\overline{f_{I,a}(T)}$. We thus obtain the last equation, that is, $f_{I,a,X}(T) = Tf_{I,a,1}(T) - d_T(f_{I,a})\overline{f_{I,a}(T)} - af_{I,a,Y}(T)$. □

4.3 RUR bitsize

We prove here, in Proposition 4.3.1, a new bound on the bitsize of the coefficients of the polynomials of a RUR. This bound is interesting in its own right and is instrumental for our analysis of the complexity of computing isolating boxes of the solutions of the input system in Section 6.1.We state our bound for RUR-*candidates*, that is even when the linear form $X + aY$ is not separating. We only use this result when the form is separating, for proving Theorem 4.1.1, but the general result is interesting in a probabilistic context when a RUR-candidate is computed with a random linear form as done in Chapter 5. We also prove our bound, not only for the RUR-candidates of an ideal defined by *two* polynomials P and Q, but for any ideal of $\mathbb{Z}[X,Y]$ that contains P and Q (for instance the radical of $\langle P,Q\rangle$ or the ideals obtained by decomposing $\langle P,Q\rangle$ according to the multiplicity of the solutions).

Proposition 4.3.1. *Let $P,Q \in \mathbb{Z}[X,Y]$ be two coprime polynomials of total degree at most d and maximum bitsize τ, let a be a rational of bitsize τ_a, and let J be any ideal of $\mathbb{Z}[X,Y]$ containing P and Q. The polynomials of the RUR-candidate of J associated to a have degree at most d^2 and bitsize in $\widetilde{O}(d^2\tau_a + d\tau)$. Moreover, there exists an integer of bitsize in $\widetilde{O}(d^2\tau_a + d\tau)$ such that the product of this integer with any polynomial in the RUR-candidate* *yields a polynomial with integer coefficients.*[1]

Remark 4.3.2. *Although we do not explicitly compute a non-asymptotic upper bound on the bitsize of the coefficients of the RUR-candidate, such an explicit bound, in $\widetilde{O}(d^2\tau_a + d\tau)$, can easily be computed following the proof of Proposition 4.3.1 and using known upper bounds on the bitsize of the resultant.*

Before proving Proposition 4.3.1, we prove a corollary of Mignotte's theorem (Theorem 2.3.9) stating that the bitsize of a factor of a polynomial P with integer coefficients does not differ much than that of P. We also recall a notion of primitive part for polynomials in $\mathbb{Q}[X,Y]$ and some of its properties.

Lemma 4.3.3 (Mignotte)**.** *Let $P \in \mathbb{Z}[X,Y]$ be of degree at most d in each variable with coefficients bitsize at most τ. If $P = Q_1Q_2$ with Q_1, Q_2 in $\mathbb{Z}[X,Y]$, then the bitsize of Q_i, $i = 1, 2$, is in $\widetilde{O}(d + \tau)$.*

Proof. A polynomial can be seen as the vector of its coefficients and we denote by $\|P\|_k$ the L^k norm of P. Mignotte lemma [Mignotte 1989, Theorem 4bis p. 172] states that $\|Q_1\|_1\|Q_2\|_1 \leqslant 2^{2d}\|P\|_2$. One always has $\|Q_i\|_\infty \leqslant \|Q_i\|_1$ and since the polynomials have integer coefficients, $1 \leqslant \|Q_i\|_\infty$. Thus $\|Q_j\|_\infty \leqslant 2^{2d}\|P\|_2$ and $\log\|Q_j\|_\infty \leqslant 2d + \log\|P\|_2$. Thus, by definition, the bitsize of Q_j is $\lfloor\log\|Q_j\|_\infty\rfloor + 1 \leqslant 2d + 1 + \log\|P\|_2$. Since P has degree at most d in each variable, it has at most $(d + 1)^2$ coefficients which are bounded by 2^τ, thus $\|P\|_2 < \sqrt{(d + 1)^2 2^{2\tau}}$ which yields that the bitsize of Q_j is less than $2d + 1 + \log(d + 1) + \tau$. □

Primitive part. Consider a polynomial P in $\mathbb{Q}[X,Y]$ of degree at most d in each variable. It can be written $P = \sum_{i,j=0}^{d} \frac{a_{ij}}{b_{ij}} X^iY^j$ with a_{ij} and b_{ij} coprime in $\mathbb{Z}$ for all i, j. We define the *primitive part* of P, denoted $pp(P)$, as P divided by the gcd of the a_{ij} and multiplied by the least common multiple (lcm) of the b_{ij}. (Note that this definition is not entirely standard since we do not consider contents that are polynomials in X or in Y.) We also denote by τ_P the bitsize of P (that is, the maximum bitsize of all the a_{ij} and b_{ij}). We prove three properties of the primitive part which will be useful in the proof.

Lemma 4.3.4. *For any two polynomials P and Q in $\mathbb{Q}[X,Y]$, we have the following properties: (i) $pp(PQ) = pp(P)\,pp(Q)$. (ii) If P is monic then $\tau_P \leqslant \tau_{pp(P)}$ and, more generally, if P has one coefficient, ξ, of bitsize τ_ξ, then $\tau_P \leqslant \tau_\xi + \tau_{pp(P)}$. (iii) If P has coefficients in $\mathbb{Z}$, then $\tau_{pp(P)} \leqslant \tau_P$.*

---

[1] In other words, the mapping $\gamma \mapsto \left(\frac{f_{J,a,X}}{f_{J,a,1}}(\gamma), \frac{f_{J,a,Y}}{f_{J,a,1}}(\gamma)\right)$ sending the solutions of $f_{J,a}(T)$ to those of J (see Definition 2.2.1) can be defined with polynomials with integer coefficients of bitsize $\widetilde{O}(d^2\tau_a + d\tau)$. This will be needed in Chapter 6.

Proof. Gauss Lemma states that if two univariate polynomials with integer coefficients are primitive, so is their product. This lemma can straightforwardly be extended to be used in our context by applying a change of variables of the form $X^iY^j \to Z^{ik+j}$ with $k > 2\max(d_Y(P), d_Y(Q))$. Thus, if P and Q in $\mathbb{Q}[X,Y]$ are primitive (i.e., each of them has integer coefficients whose common gcd is 1), their product is primitive. It follows that $pp(PQ) = pp(P)\,pp(Q)$ because, writing $P = \alpha\,pp(P)$ and $Q = \beta\,pp(Q)$, we have $pp(PQ) = pp(\alpha\,pp(P)\,\beta\,pp(Q)) = pp(pp(P)\,pp(Q))$ which is equal to $pp(P)\,pp(Q)$ since the product of two primitive polynomials is primitive.

Second, if $P \in \mathbb{Q}[X,Y]$ has one coefficient, ξ, of bitsize τ_ξ, then $\tau_P \leqslant \tau_\xi + \tau_{pp(P)}$. Indeed, We have $P = \xi\frac{P}{\xi}$ thus $\tau_P \leqslant \tau_\xi + \tau_{\frac{P}{\xi}}$. Since $\frac{P}{\xi}$ has one of its coefficients equal to 1, its primitive part is $\frac{P}{\xi}$ multiplied by an integer (the lcm of the denominators), thus $\tau_{\frac{P}{\xi}} \leqslant \tau_{pp(\frac{P}{\xi})}$ and $pp(\frac{P}{\xi}) = pp(P)$ by definition, which implies the claim.

Third, if P has coefficients in $\mathbb{Z}$, then $\tau_{pp(P)} \leqslant \tau_P$ since $pp(P)$ is equal to P divided by an integer (the gcd of the integer coefficients). □

The idea of the proof of Proposition 4.3.1 is, for $J \supseteq I = \langle P, Q\rangle$, to first argue that polynomial f_J, that is the first polynomial of the RUR-candidate before specialization at $S = a$, is a factor of f_I which is a factor of the resultant $R(T,S)$ by Lemma 4.2.5. We then derive a bound of $\widetilde{O}(d^2 + d\tau)$ on the bitsize of f_J from the bitsize of this resultant using Lemma 4.3.3. The bound on the bitsize of the other polynomials of the non-specialized RUR-candidate of J follows from the bound on f_J and we finally specialize all these polynomials at $S = a$ which yields the result. We decompose this proof in two lemmas to emphasize that, although the bound on the bitsize of f_J uses the fact that J contains polynomials P and Q, the second part of the proof only uses the bound on f_J.

Lemma 4.3.5. *Let $P, Q \in \mathbb{Z}[X,Y]$ be two coprime polynomials of total degree at most d and maximum bitsize τ, and J be any ideal of $\mathbb{Z}[X,Y]$ containing P and Q. Polynomials $f_J(T,S)$ (see (4.2)) and its primitive part have bitsize in $\widetilde{O}(d^2 + d\tau)$ and degree at most d^2 in each variable.*

Proof. Consider an ideal J containing $I = \langle P, Q\rangle$. Counted with multiplicity, the set of solutions of J is a subset of those of I thus, by Equation (4.2), polynomial $f_J(T,S)$ is monic in T and $f_J(T,S)$ divides $f_I(T,S)$. Furthermore, $f_I(T,S)$ divides $R(T,S)$ by Lemma 4.2.5. Thus $f_J(T,S)$ divides $R(T,S)$ and we consider $h \in \mathbb{Q}[T,S]$ such that $f_J\,h = R$. Taking the primitive part, we have $pp(f_J)\,pp(h) = pp(R)$ by Lemma 4.3.4. The bitsize of $pp(R)$ is in $\widetilde{O}(d^2 + d\tau)$ because R is of bitsize $\widetilde{O}(d^2 + d\tau)$ (Lemma 3.2.1) and, since R has integer coefficients, $\tau_{pp(R)} \leqslant \tau_R$ (Lemma 4.3.4). This implies that $pp(f_J)$ also has bitsize in $\widetilde{O}(d^2 + d\tau)$ by Lemma 4.3.3 because the degree of $pp(R)$ is in $O(d^2)$ (Lemma 3.2.1). Furthermore, since $f_J(T, S)$ is monic in T, $\tau_{f_J} \leqslant \tau_{pp(f_J)}$ (Lemma 4.3.4) which implies that both f_J and its primitive part have bitsize in $\widetilde{O}(d^2 + d\tau)$. Finally, the number of solutions (counted with multiplicity) of $\langle P, Q\rangle$ is at most d^2 by the Bézout bound, and this bound also holds for $J \supseteq \langle P, Q\rangle$. It then follows from Equation (4.2) that f_J has degree at most d^2 in each variable. $\square$

Lemma 4.3.6. *Let J be any ideal such that polynomials $f_J(T, S)$ (see (4.2)) and its primitive part have degree $O(d^2)$ and bitsize in $\widetilde{O}(d^2 + d\tau)$ and a is a rational of bitsize τ_a. Then all the polynomials of the RUR-candidate $RUR_{J,a}$ have bitsize in $\widetilde{O}(d^2\tau_a + d\tau)$. Moreover, there exists an integer of bitsize in $\widetilde{O}(d^2\tau_a + d\tau)$ such that its product with any polynomial in the RUR-candidate yields a polynomial with integer coefficients.*

Proof. Bitsize of $f_{J,v}$, $v \in \{1, Y\}$. We consider the equations of Lemma 4.2.3 which can be written as $\frac{\partial f_J}{\partial u}(T, S) = g_J(T, S) f_{J,v}(T, S)$ where u is T or S, and v is 1 or Y, respectively. We first bound the bitsize of one coefficient, ξ, of $f_{J,v}$ so that we can apply Lemma 4.3.4 which states that $\tau_{f_{J,v}} \leqslant \tau_\xi + \tau_{pp(f_{J,v})}$. We consider the leading coefficient ξ of $f_{J,v}$ with respect to the lexicographic order (T, S). Since g_J is monic in T (see Lemma 4.2.3), the leading coefficient (with respect to the same ordering) of the product $g_J f_{J,v} = \frac{\partial f_J}{\partial u}$ is ξ which thus has bitsize in $\widetilde{O}(\tau_{f_J})$ (since it is bounded by τ_{f_J} plus the log of the degree of f_J). It thus follows from the hypothesis on τ_{f_J} that $\tau_{f_{J,v}}$ is in $\widetilde{O}(d^2 + d\tau + \tau_{pp(f_{J,v})})$.

We now take the primitive part of the above equation (of Lemma 4.2.3), which gives $pp(\frac{\partial f_J}{\partial u}(T, S)) = pp(g_J(T, S))\ pp(f_{J,v}(T, S))$. By Lemma 4.3.3, $\tau_{pp(f_{J,v})}$ is in $\widetilde{O}(d^2 + \tau_{pp(\frac{\partial f_J}{\partial u})})$. In order to bound the bitsize of $pp(\frac{\partial f_J}{\partial u})$, we multiply $\frac{\partial f_J}{\partial u}$ by the lcm of the denominators of the coefficients of f_J, which we denote by lcm_{f_J}. Multiplying by a constant does not change the primitive part and $\mathrm{lcm}_{f_J} \frac{\partial f_J}{\partial u}$ has integer coefficients, so the bitsize of $pp(\frac{\partial f_J}{\partial u}) = pp(\mathrm{lcm}_{f_J} \frac{\partial f_J}{\partial u})$ is thus at most that of $\mathrm{lcm}_{f_J} \frac{\partial f_J}{\partial u}$ which is bounded by the sum of the bitsizes of lcm_{f_J} and $\frac{\partial f_J}{\partial u}$. By hypothesis, the bitsize of f_J is in $\widetilde{O}(d^2 + d\tau)$ so the bitsize of $\frac{\partial f_J}{\partial u}$ is also in $\widetilde{O}(d^2 + d\tau)$. On the other hand, since f_J is monic (in T), $f_J\ \mathrm{lcm}_{f_J} = pp(f_J)$ and $\tau_{\mathrm{lcm}_{f_J}} \leqslant \tau_{pp(f_J)}$ which is in $\widetilde{O}(d^2 + d\tau)$ by hypothesis. It follows that $\tau_{pp(f_{J,v})}$ and $\tau_{f_{J,v}}$ are also in $\widetilde{O}(d^2 + d\tau)$ for $v \in \{1, Y\}$.

Bitsize of $f_{J,X}$. We obtain the bound for $f_{J,X}$ by symmetry. Similarly as we proved that $f_{J,Y}$ has bitsize in $\widetilde{O}(d^2 + d\tau)$, we get, by exchanging the role of X and Y in Equation (4.2) and Lemma 4.2.3, that $\sum_{\sigma \in V(J)} \mu_J(\sigma) X(\sigma) \prod_{\varsigma \in V(J), \varsigma \neq \sigma} (T - Y(\varsigma) - SX(\varsigma))$ has bitsize in $\widetilde{O}(d^2 + d\tau)$. This polynomial is of degree $O(d^2)$ in T and S, by hypothesis, thus after replacing S by $\frac{1}{S}$ and then T by $\frac{T}{S}$, the polynomial is of degree $O(d^2)$ in T and

$\frac{1}{S}$. We multiply it by S to the power of $\frac{1}{S}$ and obtain $f_{J,X}$ which is thus of bitsize $\widetilde{O}(d^2 + d\tau)$.

Specialization at $S = a$. To bound the bitsize of the polynomials of $RUR_{J,a}$ (Definition 2.2.1), it remains to evaluate the polynomials f_J and $f_{J,v}$, $v \in \{1, X, Y\}$, at the rational value $S = a$ of bitsize τ_a. Since these polynomials have degree in S in $O(d^2)$ and bitsize in $\widetilde{O}(d^2 + d\tau)$, it is straightforward that their specializations at $S = a$ have bitsize in $\widetilde{O}(d^2 + d\tau + d^2\tau_a) = \widetilde{O}(d^2\tau_a + d\tau)$.

The lcm *of the denominators of all the coefficients in the polynomials of $RUR_{J,a}$ has bitsize $\widetilde{O}(d^2\tau_a + d\tau)$.* We have already argued that lcm_{f_J}, the lcm of the denominators of the coefficients of f_J, is in $\widetilde{O}(d^2 + d\tau)$. For each of the other polynomials $f_{J,v}$, $v \in \{1, X, Y\}$, denote by $\mathrm{lcm}_{f_{J,v}}$ and $\mathrm{gcd}_{f_{J,v}}$ the lcm of the denominators of its coefficients and the gcd of their numerators. By definition, $pp(f_{J,v}) = \frac{\mathrm{lcm}_{f_J}}{\mathrm{gcd}_{f_{J,v}}} f_{J,v}$. Let c be any coefficient of $pp(f_{J,v}) \in \mathbb{Z}[S, T]$ and $\frac{a}{b}$ be the corresponding coefficient of $f_{J,v} \in \mathbb{Q}[S, T]$ (with a and b coprime integers); we have $\mathrm{lcm}_{f_J} = c\,\frac{b}{a}\,\mathrm{gcd}_{f_{J,v}} \leqslant c\,b$ since $\mathrm{gcd}_{f_{J,v}}$ divides a. It follows that $\tau_{\mathrm{lcm}_{f_J}} \leqslant \tau_{pp(f_{J,v})} + \tau_{f_{J,v}}$ which are both in $\widetilde{O}(d^2 + d\tau)$, as proved above. Hence the lcm of the denominators of all the coefficients in $RUR_{J,a}$ has bitsize $\widetilde{O}(d^2 + d\tau)$. Finally, since all these polynomials have degree $O(d^2)$, when specializing by $S = a$, the bitsize of the denominators of the coefficients of the polynomials increase by at most $O(d^2\tau_a)$ and thus the bitsize of their lcm also increases by at most $O(d^2\tau_a)$, which concludes the proof. ☐

Proof of Proposition 4.3.1. By Lemma 4.3.5, f_J has degree at most d^2 in each variable, so has $f_{J,v}$, $v \in \{1, X, Y\}$ by Equation (4.2). It follows from Equation (4.1) that all the polynomials of any RUR-candidate of J have degree at most d^2. The rest of the proposition is a corollary of Lemmas 4.3.5 and 4.3.6. ☐

4.4 Conclusion

We presented in this chapter an algorithm for computing the Rational Univariate Representation of systems defined by two bivariate polynomials with integer coefficients. We first showed that the polynomials of the RUR of a system of two polynomials can be expressed by simple formulas which yield a new simple method for computing the RUR and also yield a new bound on the bitsize of these polynomials. This new bound implies, in particular, that the total space complexity of such RURs is, in the worst case, $\Theta(d)$ smaller than the alternative rational parametrization introduced by Gonzalez-Vega and El Kahoui [Gonzalez-Vega 1996]. Our complexity analysis shows that the cost of our new algorithm for computing a RUR is dominated by that of finding a separating form which is in $\widetilde{O}_B(d^8 + d^7\tau)$ according to the previous chapter.

Actually, the algorithm presented above is more of a theoretical than a practical interest. Indeed, the strategy that consists in computing deterministically a separating linear form is time consuming in practice, where a random choice of a linear form turns out to be almost always valid. In addition, the computation of the resultant $R(T, S)$ of trivariate polynomials is also not very efficient in practice. One particular problem of interest is thus the design of an algorithm for computing RURs of bivariate systems that is efficient in practice and whose theoretical complexity is, ideally, good as well. This is the topic of the next chapter in which we present randomized algorithms that are both efficient in practice and whose *average* theoretical complexities are substantially better than the (worst-case) complexity of the algorithm we have presented in this chapter.

Chapter 5

Efficient Practical Algorithm

Contents

As mentioned at the end of the previous chapter, the strategy that consists in computing a Rational Univariate Representation using the algorithm of Chapter 3 for computing a separating linear form and the algorithm of Chapter 4 for computing the polynomials of the RUR is not efficient in practice. First because searching deterministically for a separating linear form is time consuming, and second because the computation with polynomials in three variables is not efficient in practice. To avoid the cost of searching deterministically for a separating linear form, another strategy is to select randomly a candidate separating linear form, compute the RUR-candidate associated to this linear form and finally check that the latter is indeed a RUR (see e.g. [Rouillier 1999, Niang Diatta 2008]). This strategy transfers the difficulty of finding a separating linear form into the problem of testing whether a RUR-candidate is actually a RUR. Note that, although this strategy is natural, the latter problem does not appear to be clearly easier than the former. On the other hand, rather than handling polynomials in three variables for computing a RUR-candidate, one can opt for an alternative strategy for computing the RUR that performs a triangular decomposition based on bivariate subresultant sequences as done in [Gonzalez-Vega 1996].[1]

In this chapter, based on the two previous remarks, we investigate several algorithms for computing Rational Univariate Representations of bivariate systems. We show in particular how the use of probabilistic methods based on multi-modular strategies and random choices of separating linear forms can improve the practical efficiency, and the theoretical one in a Monte-Carlo setting and also Las-Vegas, of the RUR computation.

Our first result is a probabilistic Monte-Carlo algorithm with an expected bit complexity in $\widetilde{O}_B(d^6 + d^5\tau)$ for computing a decomposition of a bivariate system $S = \{P, Q\}$ into a set of RUR-candidates. We then propose two methods to test the result of this algorithm for correctness. While the first method is intuitively simple and conceptually elegant, the second achieves a better complexity bound. These methods yields two Las-Vegas algorithms for computing a decomposition of S into a set of RURs whose expected bit complexities are respectively in $\widetilde{O}_B(d^7+d^6\tau)$ and $\widetilde{O}_B(d^6+d^5\tau)$. Note furthermore that the former Las-Vegas algorithm is efficient in practice as demonstrated in Chapter 7, while the latter algorithm, more recent, has not yet been implemented.

A very preliminary version of this chapter was presented at the 27th European Workshop on Computational Geometry (EuroCG'11) [Bouzidi 2011].

5.1 Introduction

We first present in Section 5.3, a non-modular algorithm that, given a linear form, computes a decomposition of the initial system into a set of RUR-candidates associated to this linear form. To compute this decomposition, we proceed in two steps.

First, we decompose the initial system S into a set of triangular systems (of the form $\{A(x), B(x, y)\}$) using the subresultant sequence of P and Q. This decomposition is essentially identical to that of Gonzalez-Vega and El Kahoui [Gonzalez-Vega 1996] (see Algorithm 2 in Section 5.3) except that here, we describe how the solutions that lie on common vertical asymptotes of P and Q can be handled. Note that our approach for handling such solutions is similar to the one in [Li 2011].

In a second step, for each triangular system resulting from the previous decomposition, we compute the associated RUR-candidate using the classical

---

[1] In the paper in question, the authors consider a bivariate system defined by a polynomial and one of its partial derivatives, however the presented approach can be trivially adapted to a general bivariate system $\{P, Q\}$ as it is done for example in [Diochnos 2009].

algorithm from [Rouillier 1999].

A first interesting property that stems from the previous algorithm concerns the multiplicity of the solutions in the resulting RURs. Indeed, we prove (under the hypothesis that the chosen linear form is separating) that the multiplicity of a solution (α, β) given by a RUR computed by our algorithm is equal to the multiplicity of β in the polynomial $\gcd(P(\alpha, Y), Q(\alpha, Y))$ which we refer to as the multiplicity of (α, β) in the fiber α (see Definition 5.2.1). This property is a key feature in our application for computing the topology of curves, presented in the next chapter.

In addition, it should be stressed that even though these two algorithms (the triangular decomposition and the RUR-candidate algorithm) are standard, we show that their combination yields an improvement in terms of the worst-case arithmetic complexity for the computation of the RUR-candidates of S. More precisely, we show that computing a RUR-candidate from a triangular system resulting from the triangular decomposition can be achieved in $\widetilde{O}(D^2)$ arithmetic operations instead of $O(D^4)$ (and $\Omega(D^3)$) [Rouillier 1999], where D is the number of complex roots of the triangular system. Globally, we show that the overall computation of the RUR-candidates takes $\widetilde{O}(d^4)$ arithmetic operations, where d is the maximum total degree of P and Q, while the arithmetic complexity of the algorithm of [Rouillier 1999] is in $O(d^8)$ and $\Omega(d^6)$ in the worst case.

We however stress that the above complexity improvement should be taken with care when considering practical efficiency. Indeed, it does not necessarily imply practical improvements when performing operations in $\mathbb{Q}$ because of the possible growth of coefficients in the intermediate computations which increases the bit-complexity of arithmetic operations. This is indeed an issue in the above non-modular version of our algorithm because the coefficients of the intermediate monic triangular subsystems (or equivalently lexicographic Gröbner bases) are much larger than those of the output. However, as for many other algorithms in computer algebra, one can hope to speed up the computations in practice by using the classical framework of multi-modular arithmetic based on the Chinese Remainder Theorem (see Section 2.5).

In Section 5.4, we discuss different algorithms that stem from the use of multi-modular approach on the above algorithm. More precisely, we first apply in Section 5.4.2 a multi-modular strategy to compute the RUR-candidate of a regular triangular system $\{A(X), B(X, Y)\}$ (Algorithm 9). We then extend in Section 5.4.3 this modular algorithm to handle a general system $S = \{P, Q\}$. This yields a Monte-Carlo algorithm (Algorithm 10) of expected bit complexity in $\widetilde{O}_B(d^6 + d^5\tau)$ and whose probability of success is given in Theorem 5.4.10

In addition, in Section 5.4.4, we propose two methods for checking that the computed RUR-candidates are actually RURs of our input system. The first one is quite intuitive, it proceeds by verifying that the solutions of the RUR-candidates computed by the Monte-Carlo algorithm are solutions of the initial system with the right multiplicities. This method uses a key result which informally speaking, shows that in a multi-modular setting, under mild conditions, wrong solutions of the system can be computed but that, otherwise, no true solutions are missed. The second method is more subtle, and proceeds in two phases: it first checks that the RUR-candidates computed using a multi-modular approach are correct, and then, it checks that the chosen linear form is separating. For checking the separation, we use the strategy presented in [Diatta 2009] for testing the genericity of a curve with a slight adaptation to the case of a general bivariate system.

Hence, using the two previous tests to verify the RUR-candidates (and considering more prime numbers and linear forms until the tests succeed), we obtain from our Monte-Carlo algorithm two Las-Vegas algorithms for computing the set of Rational Univariate Representations of the input system whose expected complexities are respectively in $\widetilde{O}_B(d^7 + d^6\tau)$ and $\widetilde{O}_B(d^6 + d^5\tau)$.

5.2 Preliminaries

Multiplicities. 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. We recall this classical, though non-trivial, notion. We also introduce a simple notion of *multiplicity in fibers* that will be output by our solver and that are relevant for the topology of a plane curve.

Definition 5.2.1. *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 $\langle x-\alpha, y-\beta\rangle$. 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** *[Cox 2005, §4.2].*

We call the **fiber** *of a point* $\mathbf{p} = (\alpha,\beta)$ *the vertical line of equation $x = \alpha$. The* **multiplicity of p in its fiber** *with respect to a system of polynomials $\{P, Q\}$ in $\mathbb{Q}[x,y]$ is the multiplicity of β in the univariate polynomial* $\gcd(P(\alpha,y),\ Q(\alpha,y))$. *(This multiplicity is zero if P or Q does not vanish* *at* $\mathbf{p}$*.)*[2]

The **multiplicity of p in its fiber** *with respect to a polynomial* $P \in \mathbb{Q}[x, y]$*, denoted as* $mult(P(\alpha, y), \beta)$*, is the multiplicity of* β *in the univariate polynomial* $P(\alpha, y)$*. This multiplicity is the multiplicity of* $\mathbf{p}$ *in its fiber with respect to* $\{P, \frac{\partial P}{\partial y}\}$ *plus one.*

In Section 5.4 we use a variant of the RUR introduced in Definition 2.2.1 which we refer to as the non-reduced RUR or NRUR (note that the same definition can be found in [Alonso 1996] under the name of ARUR for, Antique Rational Univariate Representation).

Definition 5.2.2. *The NRUR-candidate of* I *associated to* $x + ay$*, denoted* $NRUR_{I,a}$ *is the following set of four univariate polynomials in* $\mathbb{Q}[T]$

$$\begin{aligned} f_{I,a}(T) &= \prod_{\sigma \in V(I)} (T - x(\sigma) - ay(\sigma))^{\mu_I(\sigma)} \\ \widehat{f}_{I,a,v}(T) &= g_{I,a}(T)\, f_{I,a,v}(T) \quad \text{for } v \in \{1, x, y\} \end{aligned} \tag{5.1}$$

where $g_{I,a}(T) = \gcd(f_{I,a}(T), \frac{\partial f_{I,a}}{\partial T}(T)) = \prod_{\sigma \in V(I)} (T - x(\sigma) - ay(\sigma))^{\mu_I(\sigma)-1}$*, and* $f_{I,a,v}(T)$ *refers to the polynomials of the RUR-candidate.*

Remark 5.2.3. *The polynomials of the NRUR-candidate consist of products between polynomials of the RUR-candidate or some of their factors, thus by Mignotte's lemma (Theorem 2.3.9), the results stated in Proposition 4.3.1 and Remark 4.3.2 about the bitsize of the polynomials of the RUR-candidate also hold for those of the NRUR-candidate.*

We state a classical property of the gcd and the gcd-free part under specialization modulo μ, which will be used in Section 5.4. Given two polynomials $A, B \in \mathbb{Z}[x]$, we denote by G the gcd of A and B and by D the gcd-free part of A with respect to B. Similarly, we denote by G^μ, D^μ their counterparts with respect to A_μ and B_μ.

Lemma 5.2.4. *Let* $A, B \in \mathbb{Z}[x]$ *and* μ *be a prime number such that* $\phi_\mu(A) \not\equiv 0$ *or* $\phi_\mu(B) \not\equiv 0$*. Then,* $\phi_\mu(G)$ *divides* G^μ *and* D^μ *divides* $\phi_\mu(D)$*.*

Proof. Writing $A = G \times D$ and $B = G \times H$ and applying the reduction modulo μ on both sides of the two equations, we obtain $A_\mu = \phi_\mu(G) \times \phi_\mu(D)$ and $B_\mu = \phi_\mu(G) \times \phi_\mu(H)$. The polynomial $\phi_\mu(G)$ is not zero in $\mathbb{Z}_\mu$ since $\phi_\mu(A) \not\equiv 0$ or $\phi_\mu(B) \not\equiv 0$ and it divides both A_μ and B_μ, thus, it also divides their gcd G^μ. On the other hand, writing $A_\mu = G^\mu \times D^\mu$, we have $\phi_\mu(G) \times \phi_\mu(D) = G^\mu \times D^\mu$. Since $\phi_\mu(G) \not\equiv 0$ we can divide both sides of this equation by $\phi_\mu(G)$, to obtain $\phi_\mu(D) = \frac{G^\mu}{\phi_\mu(G)} \times D^\mu$ and the second claim follows since $\phi_\mu(G)$ divides G^μ. □

---

[2] The gcd is considered over $\mathbb{Q}(\alpha)[y]$, the ring of polynomials in y with coefficients in the field extension of $\mathbb{Q}$ by α.

5.3 School-book non-modular algorithm

The algorithm presented in this section first decomposes the system $S = \{P, Q\}$ into a set of triangular systems using the triangular decomposition of Gonzalez-Vega and El Kahoui [Gonzalez-Vega 1996] which we recalled in Section 3.5.1. This triangular decomposition algorithm is actually slightly adapted here in order to handle the case of a system S having solutions that lie on a common vertical asymptote of P and Q, similarly as in [Li 2011]. Then, for each triangular system resulting from this decomposition, the algorithm computes the associated RUR-candidates using the classical algorithm from [Rouillier 1999]. The output of our non-modular algorithm is thus a set of RUR-candidates, which encode all the solutions of S if the chosen candidate linear separating form is indeed separating.

5.3.1 Triangular decomposition

The first step of our algorithm consists in decomposing the input system $S = \{P, Q\}$ into a set of triangular systems of the form $\{A(x), B(x, y)\}$. This decomposition is identical to that of Algorithm 2 except that here, we treat the case where some solutions lie on some common vertical asymptotes of the curves defined by P and Q. For completeness, we recall in Algorithm 6, the triangular decomposition that handles the solutions away from common asymptotes and then apply the latter recursively in Algorithm 7 in order to obtain all the solutions of $\{P, Q\}$. We moreover show that the multiplicities of the solutions in the obtained triangular systems correspond to the multiplicities of these solutions in their fibers for the system $\{P, Q\}$ (see Definition 5.2.1). As indicated in the beginning of this chapter, this is important in our application for computing the topology of curves, presented in the next chapter.

Output. The output of Algorithm 7 consists of a set of triangular systems $S_k = \left\{ \begin{array}{l} F_k(x) = 0 \\ Sres_k(x, y) = 0 \end{array} \right.$ such that $\gcd(F_k(x),\, sres_k(x)) = 1$ and all F_k are squarefree. Note that for every k there might be several systems S_k depending on whether the solution of S_k lies on a vertical asymptote or not; $Sres_k$ denotes the k-th subresultant of P and Q, or of reductions of P and Q modulo polynomials in $\mathbb{Q}[x]$ that divide at least one of their leading coefficients. In addition, we prove the following result on multiplicities.

Lemma 5.3.1. *The multiplicity of a solution in S_k is the multiplicity in its fiber for the system $\{P, Q\}$.*

Algorithm 6 Triangular decomposition away from asymptotes [Gonzalez-Vega 1996, Li 2011]

```
Require: $P, Q$ in $\mathbb{Z}[x,y]$ defining a finite set of solutions in $\mathbb{C}^2$, $A$ in $\mathbb{Z}[x]$.
Ensure: $\mathcal{TD}$ a set of triangular systems $\{F_i(x), Sres_i(x,y)\}_{i\in\mathcal{I}}$ describing the
    solutions of $\{P, Q\}$ away from asymptotes, and whose $x$-coordinates are
    roots of $A$.
 1: Compute the subresultant sequence of $P$ and $Q$
 2: $F = Fres(P,Q,A) = \gcd(\frac{\overline{Res_y(P,Q)}}{\overline{\gcd(Lc(P),Lc(Q))}}, A)$
 3: $G_0 = F$
 4: $\mathcal{TD} = \emptyset$
 5: for $i = 1$ to $\mathrm{d}_y(Q)$ do
 6:     $G_i = \gcd(G_{i-1}, sres_i(P,Q))$
 7:     $F_i = G_{i-1}/G_i$
 8:     if $d_x(F_i) > 0$, add $S_i = \{F_i, Sres_i(P,Q)\}$ to $\mathcal{TD}$
 9: end for
10: return $\mathcal{TD}$
```

Proof. Since $F_k(x)$ a squarefree polynomial, the multiplicity of any solution (α,β) in $S_k = \{F_k(x), Sres_k(x,y)\}$ is equal to the multiplicity of β in the univariate polynomial $Sres_k(\alpha,y)$. This is also equal to the multiplicity of (α,β) in its fiber for the system $\{P,Q\}$ since $Sres_k(\alpha,y) = \gcd(P(\alpha,y), Q(\alpha,y))$ by Theorem 2.4.15. ☐

The following lemma is critical for bounding the degree and the bitsize of the RUR-candidates we compute in this chapter.

Lemma 5.3.2. *Let I_k be the ideal associated to a triangular system S_k. The ideal $I = \langle P,Q\rangle$ is contained in I_k. Equivalently, the multiplicity of a solution in S_k is smaller than or equal to its multiplicity in $\langle P,Q\rangle$.*

Proof. We first perform the pseudo-division of $P(x,y)$ by $Sres_k(x,y)$ with respect to the variable y. This yields two polynomials $U, R \in \mathbb{Z}[x,y]$ such that

$$sres_k^{\delta}(x)P(x,y) = U(x,y)Sres_k(x,y) + R(x,y),$$

where $\delta \in \mathbb{N}$ and where $\deg_y(R(x,y)) < \deg_y(Sres_k(x,y))$.

In the following, we denote by α a root of $F_k(x)$. Replacing x by α in the above equation gives

$$sres_k^{\delta}(\alpha)P(\alpha,y) = U(\alpha,y)Sres_k(\alpha,y) + R(\alpha,y).$$

Furthermore, $\deg_y(R(\alpha,y)) < \deg_y(Sres_k(\alpha,y))$ because, by definition of F_k (see Algorithm 6, Line 7), F_k is coprime with the leading coefficient $sres_k(x)$

Algorithm 7 Triangular Decomposition [Li 2011]

```
Require: $P, Q$ in $\mathbb{Z}[x, y]$ defining a finite set of solutions in $\mathbb{C}^2$.
Ensure: $\mathcal{TD}$ a set of triangular systems $\{F_i(x), Sres_i(x, y)\}_{i \in \mathcal{I}}$ describing the
    set of solutions of the system $\{P, Q\}$.
$\mathcal{TD} = \{\}$
$A = 0$
repeat
    $\mathcal{TD} = \mathcal{TD} \cup$ Algorithm 6$(P, Q, A)$
    $A = \overline{\gcd(A, \mathrm{Lc}_y(P), \mathrm{Lc}_y(Q))}$
    $P = P \mod A$, $Q = Q \mod A$
until $A \in \mathbb{Q}$
return $\mathcal{TD}$
```

of $Sres_k(x, y)$ (seen as a polynomial in y) and thus $\deg_y(Sres_k(\alpha, y)) = deg_y(Sres_k(x, y))$.

On the other hand, by Lemma 3.5.1, $Sres_k(\alpha, y)$ is a gcd of $P(\alpha, y)$ and $Q(\alpha, y)$ and thus, there exists $E(y) \in \mathbb{Q}(\alpha)[y]$ such that $P(\alpha, y) = E(y) Sres_k(\alpha, y)$. Substituting $P(\alpha, y)$ in the previous equation yields

$$R(\alpha, y) = [sres_k^{\delta}(\alpha) E(y) - U(\alpha, y)] Sres_k(\alpha, y).$$

Since $\deg_y(R(\alpha, y)) < \deg_y(Sres_k(\alpha, y))$, this implies that $R(\alpha, y)$ identically vanishes. This holds for any root α of $F_k(x)$, which implies that $F_k(x)$ is a factor of $R(x, y)$, or in other words, that there exists $V(x, y) \in \mathbb{Z}[x, y]$ such that $R(x, y) = F_k(x) V(x, y)$.

By substituting $R(x, y)$ in the previous pseudo-division equality, we obtain that $sres_k^{\delta}(x) P(x, y) = U(x, y) Sres_k(x, y) + V(x, y) F_k(x)$, which proves that the polynomial $sres_k^{\delta}(x) P(x, y)$ is in the ideal I_k.

Similarly, we have that $sres_k^{\delta'}(x) Q(x, y)$ is in I_k, for some $\delta' \in \mathbb{N}$, and thus the ideal $J = \langle sres_k^{\delta}(x) P(x, y), sres_k^{\delta'}(x) Q(x, y) \rangle$ is contained in the ideal I_k.

As a consequence, given a solution $\gamma = (\alpha, \beta)$ of I_k, we have that the multiplicity of γ in the ideal I_k, denoted as $mult(\gamma, I_k)$, is smaller or equal than $mult(\gamma, J)$.

On the other hand, a property of the multiplicity stated in [Fulton 1989, §3.3, Property 6] is that, for any three polynomials $A, B, C \in \mathbb{Z}[x, y]$, $mult(\gamma, \langle A, BC \rangle) = mult(\gamma, \langle A, B \rangle) + mult(\gamma, \langle A, C \rangle)$. Applied on J, this implies that $mult(\gamma, J) = mult(\gamma, I)$ since $sres_k(x)$ does not vanish at α. Hence, $mult(\gamma, I_k) \leqslant mult(\gamma, I)$, which is also equivalent to saying that the ideal $I = \langle P, Q \rangle$ is contained in I_k. □

Remark. Although in practice, our algorithm handles the case of systems having some solutions located on common vertical asymptotes, however, in order to simplify the theoretical analysis below, *we will assume in the rest of this chapter that no common vertical asymptotes exist, that is, that the leading coefficients of P and Q with respect to y are coprime.* In that case, only one iteration of the loop in Algorithm 7 is sufficient to compute the triangular decomposition of all the solutions of $\{P, Q\}$. Note that the presence of common vertical asymptotes can increases by a factor d in the worst case, the complexity of computing the above triangular decomposition, however this can be avoided by performing a change of variable that transforms the input system into a new system without common vertical asymptotes (see [Diochnos 2009] for details). In practice however, this turns out to be less efficient than the above strategy.

The following theorem provides the complexity of the triangular decomposition under the above hypothesis. The proof of the arithmetic complexity is given in Lemma 3.5.2 while that of the bit complexity can be found in [Diochnos 2009, Proposition 18].

Theorem 5.3.3. *Let $S = \{P, Q\}$ with $P, Q \in \mathbb{Z}[x, y]$ of total degree at most d and maximum bitsize τ such that $\mathrm{Lc}_y(P)$ and $\mathrm{Lc}_y(Q)$ are coprime. Algorithm 7 computes the triangular decomposition of S using $\widetilde{O}(d^4)$ arithmetic operations and $\widetilde{O}_B(d^7 + d^6\tau)$ bit operations.*

5.3.2 Rational Univariate Representation

The decomposition of the previous section yields triangular systems of the form $\mathcal{T} = \{B(x, y), A(x)\}$ with the property that $A(x)$ and $\mathrm{Lc}_y(B)(x)$ are coprime (such systems are referred to as regular triangular systems). Thus, by Bèzout's identity, there exists two polynomials $U, V \in \mathbb{Q}[x]$ such that $U\,A + V\,\mathrm{Lc}_y(B) = 1$ and the system $\mathcal{T}$ can then be rewritten as $\widetilde{\mathcal{T}} = \begin{cases} x^{d_x} + R_1(x) = 0 \\ y^{d_y} + R_2(x, y) = 0 \end{cases}$ where $y^{d_y} + R_2(x, y)$ is the reduction of $V\,B$ modulo A (that is each coefficient of $B(x, y)$ in $\mathbb{Z}[x]$ is reduced modulo $A(x)$); R_1 is of degree at most $d_x - 1$ and R_2 is of degree at most $d_y - 1$ in y and at most $d_x - 1$ in x. The resulting system $\widetilde{\mathcal{T}}$ is a (reduced) lexicographic Gröbner basis according to Buchberger's criteria [Cox 2007].

The ideal I associated to $\widetilde{\mathcal{T}}$ has a finite number of zeros which implies that the quotient algebra $\mathbb{Q}[x, y]/I$ is a finite dimensional $\mathbb{Q}$ vector space. Moreover, since the leading monomials of the polynomials in $\widetilde{\mathcal{T}}$ are x^{d_x} and y^{d_y}, $\mathbb{Q}[x, y]/I$ has dimension $D = d_x d_y$ which is also the number of zeros of I counted with multiplicities [Cox 2007] and its basis is simply $\{x^i y^j \mid 0 \leqslant i < d_x,\ 0 \leqslant j < d_y\}$ which is the set of monomials that cannot be reduced by the Gröbner basis of I [Cox 2005, §5.3].

Algorithm 8 RUR-candidate [Rouillier 1999]

```
Require: A basis of $\mathbb{Q}[x,y]/I$, $T = x + ay$ a linear form
Ensure: The RUR-candidate of $I$ associated to $T$
1: Compute the multiplication tensor of $\mathbb{Q}[x,y]/I$, $T = x + ay$
2: Compute $\mathrm{Trace}(M_{vT^i})$, the trace of the multiplication matrix by the polynomial $vT^i$ in $\mathbb{Q}[x,y]/I$ for $i = 1, \ldots, D$ and $v \in \{1, x, y\}$.
3: Solve the triangular linear system $\{(D - i)a_i = \sum_{j=0}^{i-1} a_{i-j}\mathrm{Trace}(M_{T^j})\}_{i=0,\ldots,D-1}$ and set $f_{I,a}(T) = \sum_{i=0}^{D-1} a_i T^{D-i}$
4: Compute $\overline{f_{I,a}(T)} = \sum_{i=0}^{d-1} b_i T^{d-i}$ and set $H_k = \sum_{i=0}^{k} b_i T^{k-i}$ for $k = 0, \ldots, d-1$, the $k$-th Horner polynomial associated to $\overline{f_{I,a}(T)}$
5: Compute $f_{I,a,v} = \sum_{i=0}^{d-1} \mathrm{Trace}(M_{vT^i}) H_{d-i-1}(T)$ for $v \in \{1, x, y\}$
6: return $f_{I,a}, f_{I,a,v}$ for $v \in \{1, x, y\}$
```

One way to isolate the solutions of $\widetilde{\mathcal{T}}$ while keeping the multiplicity information is to compute a Rational Univariate Representation (RUR) (Definition 2.2.1). The algorithm presented in [Rouillier 1999] for computing a RUR first considers a random linear form $x + ay$ (starting by x and y) and assumes that this linear form is *separating*. Then, it computes a RUR-candidate and checks, at the end, if it is correct; if not, another candidate separating linear form is chosen. The algorithm terminates because only $O(D^2)$ linear polynomials are *not* separating, which also implies that a random choice is good with high probability. In the following, we recall the algorithm for computing a RUR-candidate presented in [Rouillier 1999] (Algorithm 8) and show how the particular structure of the system $\widetilde{\mathcal{T}}$ decreases the worst-case arithmetic complexity from $O(D^4)$ and $\Omega(D^3)$ to $\widetilde{O}(D^2)$.[3]

The dominant part in the computation of the RUR-candidate as described in [Rouillier 1999] is the computation of the multiplication tensor of the quotient algebra $\mathbb{Q}[x,y]/I$ (i.e. all the images in $\mathbb{Q}[x,y]/I$ of all the possible products of two elements of a monomial basis of $\mathbb{Q}[x,y]/I$), which is a key element for the computation of the traces $\mathrm{Trace}(M_{vT^i})$ (Line 2). This computation is done iteratively by multiplying an already reduced product by one variable in $\mathbb{Q}[x,y]/I$ to get the reduction of another product. The cost of such a computation depends on two parameters: the number of products to be computed, denoted by K in the sequel, and the cost of the multiplication by one variable in $\mathbb{Q}[x,y]/I$. In [Rouillier 1999] the basis of $\mathbb{Q}[x,y]/I$ is given by all the monomials that cannot be reduced modulo an arbitrary Gröbner basis of I so that $K \in O(D^2)$ and the multiplication by one variable is reduced to a

---

[3] These complexities hold for *one* choice of candidate separating linear form.

matrix/vector product requiring $O(D^2)$ arithmetic operations which yields a total of $O(D^4)$ arithmetic operations for computing the multiplication tensor.

In our setting, a basis of the algebra $\mathbb{Q}[x,y]/I$ is $\{x^i y^j \mid 0 \leqslant i < d_x,\ 0 \leqslant j < d_y\}$, thus, all the products of two elements in this basis result in the set $\{x^i y^j \mid 0 \leqslant i < 2d_x,\ 0 \leqslant j < 2d_y\}$ which is of size $K = 4d_x d_y = 4D$. On the other hand, we consider the reduction modulo I of the product by x and y of an already reduced polynomial. The facts that the system $\tilde{\mathcal{T}}$ is triangular and that the leading monomial of the bivariate polynomial is pure (y^{d_y}) yield that each reduction can be performed in $\widetilde{O}(D)$ operations instead of $O(D^2)$. The reduction modulo I of the product of x by a polynomial T already reduced in $\mathbb{Q}[x,y]/I$ is

$$\begin{aligned}
\overline{x \times T(x,y)} &= \overline{\textstyle\sum_{i=0}^{d_x-1} \sum_{j=0}^{d_y-1} a_{i,j} x^{i+1} y^j} \\
&= \overline{\textstyle\sum_{i=1}^{d_x-1} \sum_{j=0}^{d_y-1} a_{i-1,j} x^i y^j} + \overline{x^{d_x} \times \textstyle\sum_{j=0}^{d_y-1} a_{d_x-1,j} y^j} \\
&= \textstyle\sum_{i=1}^{d_x-1} \sum_{j=0}^{d_y-1} a_{i-1,j} x^i y^j - R_1(x) \times \sum_{j=0}^{d_y-1} a_{d_x-1,j} y^j
\end{aligned}$$

in which the polynomial multiplication and difference can be computed in a straightforward manner with $O(d_x d_y) = O(D)$ arithmetic operations. The reduction of the multiplication by y is

$$\begin{aligned}
\overline{y \times T(x,y)} &= \overline{\textstyle\sum_{i=0}^{d_x-1} \sum_{j=0}^{d_y-1} a_{i,j} x^i y^{j+1}} \\
&= \overline{\textstyle\sum_{i=0}^{d_x-1} \sum_{j=1}^{d_y-1} a_{i,j-1} x^i y^j} + \overline{y^{d_y} \textstyle\sum_{i=0}^{d_x-1} a_{i,d_y-1} x^i} \\
&= \textstyle\sum_{i=1}^{d_x-1} \sum_{j=0}^{d_y-1} a_{i-1,j} x^i y^j - \overline{R_2(x,y) \textstyle\sum_{i=0}^{d_x-1} a_{i,d_y-1} x^i} \\
&= \textstyle\sum_{i=1}^{d_x-1} \sum_{j=0}^{d_y-1} a_{i-1,j} x^i y^j \\
&\qquad - \textstyle\sum_{k=0}^{deg_y(R_2)} \overline{\gamma_k(x) \textstyle\sum_{i=0}^{d_x-1} a_{i,d_y-1} x^i}\, y^k
\end{aligned}$$

where $\gamma_k(x)$ denotes the coefficient of y^k in $R_2(x,y)$. Then, the multiplication of $\gamma_k(x)$ and $\sum_{i=0}^{d_x-1} a_{i,d_y-1} x^i$ can be done in $\widetilde{O}(d_x)$ arithmetic operations using fast univariate polynomial multiplication (Theorem 2.3.1); this gives a univariate polynomial in x whose reduction modulo I, that is modulo the x-univariate polynomial of $\tilde{S}$, can also be done in $\widetilde{O}(d_x)$ arithmetic operations (Theorem 2.3.2). This has to be done for every $k \leqslant deg_y(R_2) < d_y$ which thus takes $\widetilde{O}(d_x d_y)$, and so $\widetilde{O}(D)$, operations in total. Hence, there are $O(D)$ reductions which take $\widetilde{O}(D)$ operations each, giving the claimed complexity of $\widetilde{O}(D^2)$.

To compute the RUR-candidate of the regular triangular system $\mathcal{T} = \{B(x,y), A(x)\}$, one still needs to compute a basis of the quotient algebra associated to $I = \langle B(x,y), A(x)\rangle$, or equivalently, the reduced lexicographic Gröbner basis of I. The following lemma gives the complexity of the computation of the RUR-candidate including the cost of this step.

Lemma 5.3.4. *Let $\mathcal{T} = \{B(x,y), A(x)\}$ be a triangular system with $d_x(A) \leqslant d_x(B)$ and $d_y(B)d_x(A) = d_{\mathcal{T}}$ and such that $Lc_y(B)$ and A are coprime. Let $T = x + ay$ be a linear form with a an integer of bitsize in $\widetilde{O}(1)$. The RUR-candidate of $\mathcal{T}$ associated to T can be computed using $\widetilde{O}(d_{\mathcal{T}}^2 + d_x(B)d_y(B))$ arithmetic operations.*

Proof. The RUR-candidate of $\mathcal{T}$ is computed in two steps. First, the triangular system $\mathcal{T}$ is transformed into a lexicographic Gröbner basis $\tilde{\mathcal{T}}$ by computing $\mathrm{Lc}_y(B)^{-1}$, the inverse of $\mathrm{Lc}_y(B)$ with respect to A, which can be done with $\widetilde{O}(d_x(B))$ arithmetic operations by [von zur Gathen 2003, Corollary 11.11] and then by computing the reduction of $\mathrm{Lc}_y(B)^{-1}\, B$ modulo A, which can be done in $\widetilde{O}(d_x(B)d_y(B))$, according to [von zur Gathen 2003, Corollary 11.8]. Second, the RUR-candidate of $\tilde{\mathcal{T}}$ is computed using Algorithm 8 which, as shown above, requires $\widetilde{O}(d_{\mathcal{T}}^2)$ arithmetic operations. □

The following result gives the arithmetic complexity for computing the RUR-candidates of all the regular triangular systems output by Algorithm 7.

Theorem 5.3.5. *Let $S = \{P, Q\}$ with $P, Q \in \mathbb{Z}[x,y]$ of total degree at most d and maximum bitsize τ and $\{S_0, S_1, \ldots, S_q\}$ be the regular triangular systems associated to S. The RUR-candidates of all the systems S_k can be computed using $\widetilde{O}(d^4)$ arithmetic operations.*

Proof. The systems resulting from the decomposition of S are of the form $S_k = \{Sres_k(x,y), F_k(x)\}$. According to Theorem 2.4.16, $d_x(Sres_k)$ as well as $d_x(F_k)$ are in $O(d^2)$ while $d_y(Sres_k)$ is in $O(d)$. We also have by Lemma 5.3.2 and Bézout's bound on the system $\{P, Q\}$ that $\sum_{k\in\{1,\ldots,q\}} d_x(F_k)d_y(Sres_k) \leqslant d^2$. By Lemma 5.3.4, the RUR-candidate of a triangular system S_k can be computed using $\widetilde{O}(d_x(F_k)^2 d_y(Sres_k)^2 + d^3)$ arithmetic operations. Summing over all the systems S_k for $k \in \{1, \ldots, q\}$ where q is in $O(d)$, we obtain the stated complexity. □

The following theorem summarizes the results obtained in this section. Its proof follows straightforwardly from Theorem 5.3.3 and the previous theorem.

Theorem 5.3.6. *Given a system $S = \{P, Q\}$ with $P, Q \in \mathbb{Z}[x,y]$ of total degree at most d such that $\mathrm{Lc}_y(P)$ and $\mathrm{Lc}_y(Q)$ are coprime and a linear form $T = x + ay$. The decomposition of S into a set of RUR-candidates can be computed using $\widetilde{O}(d^4)$ arithmetic operations.*

5.4 Modular algorithm

The size of the coefficients in a RUR-candidate is reasonable in the sense that they have the same asymptotic bound on the size, as those in the squarefree part of the resultant according to Lemma 5.3.2, Proposition 4.3.1 and Theorems 2.4.16 and 2.3.10. However, the size of the coefficients in intermediate computations of the algorithm presented in the previous section may be quite large. Typically, the transformation of the regular triangular systems into lexicographic Gröbner bases leads in general to a significant growth of the coefficients, which slows the computation of the RUR-candidates. To avoid such intermediate coefficients growth, we use a standard multi-modular strategy. Roughly speaking, this consists in computing the decomposition of the input system into RUR-candidates modulo several prime numbers and then, lifting the result over the rationals using the Chinese Remainder Algorithm.

The use of a multi-modular approach may however leads in some cases to a wrong result. In our case, this may happen if, either not enough prime numbers have been used, which leads to the failure of the lifting phase in the Chinese Remainder Algorithm, or if for some selected prime numbers μ, the RUR-candidates are not well specialized, i.e. the images of the RUR-candidates modulo μ are not equal to the RUR-candidates computed with the input system modulo μ. The first issue can be easily handled by computing a bound on the size of the coefficients of the RUR-candidates polynomials using Proposition 4.3.1 together with Remark 4.3.2 and running the algorithm on a set of prime numbers whose product is larger than this bound.[4] To handle the second problem, we first slightly modify the output of our modular algorithm. Instead of a RUR-candidates, we compute non-reduced RUR-candidates (see Definition 5.2.2) which are shown to have better specialization properties under certain conditions.

5.4.1 Overview

As a first step, in Section 5.4.2, we design Algorithm 9 which, given a regular triangular system $\mathcal{T} = \{B(x, y), A(x)\}$ and a linear form $x + ay$, computes the associated RUR-candidate using multi-modular computations. This algorithm selects a set of prime numbers μ satisfying the good specialization property of Definition 5.4.2, computes over $\mathbb{Z}_\mu$ the non-reduced RUR-candidate of $\mathcal{T}$ modulo each of these prime numbers and finally lifts the result over the rational using the Chinese Remainder Algorithm. The RUR-candidate is then computed from the non-reduced one using Definition 5.2.2.

Afterward, in Section 5.4.3, we design Algorithm 10 which extends the multi-modular strategy of Algorithm 9 to a general system $S = \{P, Q\}$. This algorithm follows the approach of Section 5.3, i.e. a triangular decomposition

---

[4] The asymptotic bound stated in Proposition 4.3.1 for the RUR-candidate of a general system $\{P, Q\}$, also holds for the RUR-candidates of the triangular systems presented above according to Lemma 5.3.2.

followed by the computation of RUR-candidates. More precisely, for a set of prime numbers the algorithm computes the triangular decomposition of S and the non-reduced RUR-candidates associated to the resulting regular triangular systems modulo each of these primes and it then lifts the resulting non-reduced RUR-candidates over the rational.

Unlike Algorithm 9, the prime numbers are now chosen randomly in a given set of primes and without guarantee on the specialization of the computed non-reduced RUR-candidates. This yields a Monte-Carlo algorithm which we analyze the expected bit complexity and the probability of success.

We finally show, in Section 5.4.4, how this Monte-Carlo algorithm can be transformed into a Las-Vegas by adding a verification that ensures the correctness of the result. More precisely, we provide two different methods for verifying that the computed RUR-candidates are indeed RURs of the input system. One of these methods uses a key result proved on the Monte-Carlo algorithm which roughly says that wrong solutions of the system can be computed but that no true solutions are missed (Lemma 5.4.9).

5.4.2 RUR-candidate for $\mathcal{T} = \{\mathbf{B}(\mathbf{x}, \mathbf{y}), \mathbf{A}(\mathbf{x})\}$

We consider a regular triangular system $\mathcal{T} = \{B(x, y), A(x)\}$, and denote by I the associated ideal. Algorithm 9 computes a RUR-candidate for $\mathcal{T}$ using multi-modular computations. It should be stressed that we present this algorithm only for pedagogical reasons as it introduces some tools used in our main algorithm presented in Section 5.4.3.

Proposition 5.4.3 is fundamental for the correctness of this algorithm: it gives conditions on a prime number μ so that the NRUR-candidate of $\mathcal{T}$ is well specialized modulo μ.

Equivalently as for the RUR-candidate (Algorithm 8), the polynomials of a NRUR-candidate can be expressed using trace formulas, as shown in the following lemma. For completeness, we provide a proof which is a straightforward adaptation of the proof of [Rouillier 1999, Theorem 3.1].

Lemma 5.4.1. *For $v \in \{1, x, y\}$, $\widehat{f}_{I,a,v} = \sum_{i=0}^{D-1} Trace(M^I_{vT^i})\, H_{D-i-1}(T)$ where $H_k(T) = \sum_{i=0}^{k} a_i\, T^{k-i}$ is the i-th Horner polynomial associated to $f_{I,a}$ and M^I_h is the matrix of the multiplication by $h \in \mathbb{Q}[x, y]$ in $\frac{\mathbb{Q}[x,y]}{I}$.*

Proof. Following the proof of [Rouillier 1999, Theorem 3.1], we first notice that $\frac{\widehat{f}_{I,a,v}}{f_{I,a}} = \sum_{\alpha \in V(I)} \frac{\mu(\alpha)v(\alpha)}{T - T(\alpha)}$, which is equal to $\sum_{i \geqslant 0} \frac{\sum_{\alpha \in V(I)} \mu(\alpha)v(\alpha)T(\alpha)^i}{T^{i+1}}$. On the other hand, $\sum_{\alpha \in V(I)} \mu(\alpha)v(\alpha)T(\alpha)^i = Trace(M^I_{vT^i})$ and thus $\frac{\widehat{f}_{I,a,v}}{f_{I,a}} = \sum_{i \geqslant 0} \frac{Trace(M^I_{vT^i})}{T^{i+1}}$. Multiplying both sides by $f_{I,a} =$

$\sum_{i=0}^{D-1} a_i T^i$, we then get $\widehat{f}_{I,a,v} = \sum_{i=0}^{D-1} \sum_{j=0}^{D-i-1} Trace(M^I_{vT^i})\, a_j T^{D-i-j-1} = \sum_{i=0}^{D-1} Trace(M^I_{vT^i})\, H_{D-i-1}(T)$. ☐

Definition 5.4.2. *Let $\mathcal{T} = \{B(x,y), A(x)\}$ be a regular triangular system. A prime number μ is said to be lucky with respect to $\mathcal{T}$ if and only if $\phi_\mu(A)$ and $\phi_\mu(\mathrm{Lc}_y(B))$ are coprime.*

Proposition 5.4.3. *If μ is lucky with respect to $\mathcal{T}$, then $f_{I^\mu,\phi_\mu(a)} = \phi_\mu(f_{I,a})$ and $\widehat{f}_{I_k^\mu,\phi_\mu(a),v} = \phi_\mu(\widehat{f}_{I,a,v})$ for $v \in \{1, x, y\}$.*

Proof. Similarly as $\widetilde{\mathcal{T}} = \{(\mathrm{Lc}_y(B))^{-1}B, A\}$ consists of a (non-reduced) Gröbner basis of I for the lexicographic ordering $x < y$ (see Section 5.3.2), when $\phi_\mu(A)$ and $\phi_\mu(\mathrm{Lc}_y(B))$ are coprime, the system $\widetilde{\mathcal{T}}^\mu = \{(\mathrm{Lc}_y(B)^\mu)^{-1}B^\mu, A^\mu\}$ is also a (non-reduced) Gröbner basis of I^μ for the lexicographic ordering $x < y$. Thus, I and I^μ are both zero-dimensional and both vector spaces $\frac{\mathbb{Q}[x,y]}{I}$ and $\frac{\mathbb{Z}_\mu[x,y]}{I^\mu}$ are generated by the same monomial basis $\{x^i y^j, i = 0 \dots d_x - 1, j = 0 \dots d_y - 1\}$.

Let $NF_{I,x<y}$ (resp. $NF_{I^\mu,x<y}$) be the (uniquely defined) normal form associated to the Gröbner basis $\widetilde{\mathcal{T}}$ (resp. $\widetilde{\mathcal{T}}^\mu$). This normal form defines the canonical surjection $\mathbb{Q}[x,y] \to \mathbb{Q}[x,y]/I$ (resp. $\mathbb{Z}_\mu[x,y] \to \mathbb{Z}_\mu[x,y]/I^\mu$). Since ϕ_μ is a morphism, $\forall h \in \mathbb{Q}[x,y]$, $\phi_\mu(NF_{I,x<y}(h)) = NF_{I^\mu,x<y}(\phi_\mu(h))$, which also implies that $\phi_\mu(M^I_h) = M^{I^\mu}_{\phi_\mu(h)}$.

It first follows that $\phi_\mu(f_{I,a}) = f_{I^\mu,\phi_\mu(a)}$ since they have the same degree and share the same Newton sums, and second, that $\phi_\mu(\widehat{f}_{I,a,v}) = \phi_\mu(\sum_{i=0}^{D-1} Trace(M^I_{vT^i})\, H_{D-i-1}(T)) = \sum_{i=0}^{D-1} Trace(M^{I^\mu}_{vT^i})\, H_{D-i-1}(T)$ which is equal to $\widehat{f}_{I^\mu,\phi_\mu(a),v}$. ☐

The correctness of Algorithm 9 follows directly from Proposition 5.4.3 and the Chinese Remainder Theorem 2.5.2, noticing that the selected prime numbers have their product larger than the size of the output.

Lemma 5.4.4. *Let $\mathcal{T} = \{B(x,y), A(x)\}$ as in Lemma 5.3.4, with $B(x,y)$, $A(x)$ of maximum bitsize τ'. Let $T = x + ay$ be a linear form with a an integer of bitsize in $\widetilde{O}(1)$. Algorithm 9 computes the RUR-candidate of $\mathcal{T}$ associated to T using $\widetilde{O}_B(m(d_x(B)d_y(B) + d_{\mathcal{T}}^2) + d_x(B)^2\tau')$ bit operations, where m denotes an upper-bound on the bitsize of the coefficients in the NRUR-candidate of $\mathcal{T}$.*

Proof. Let m be an upper bound on the bitsize of the coefficients of $f_{I,a}, \widehat{f}_{I,a,v}$, $v \in \{1, x, y\}$. In line 2, we compute a set $\mathcal{P}$ of lucky prime numbers

---

[5] Algorithm 8 is slightly modified, replacing the formulas of Line 5 by those of Lemma 5.4.1.

Algorithm 9 RUR-candidate for $\mathcal{T} = \{B(x,y), A(x)\}$

```
Require: $\mathcal{T} = \{B(x,y), A(x)\}$ a regular triangular system and $T = x + ay$ a linear form
Ensure: The RUR-candidate of $\mathcal{T}$ associated to $T$
Compute an integer $m$ such that $2^m$ is greater than the maximum size of the coefficients of $f_{I,a}, \widehat{f}_{I,a,v},\ v \in \{1,x,y\}$ (see Remark 5.2.3)
Compute a set $\mathcal{P}$ of lucky prime numbers with respect to $\mathcal{T}$ such that $\prod_{\mu\in\mathcal{P}} \mu > 2^m$
Compute $\mathcal{T}^\mu = \{\phi_\mu(B(x,y)), \phi_\mu(A(x))\}$ for all $\mu \in \mathcal{P}$
for $\mu \in \mathcal{P}$ do
    Compute the NRUR-candidate of $\mathcal{T}^\mu$ i.e. $f_{I^\mu,a}, \widehat{f}_{I^\mu,\phi_\mu(a),v}\ v \in \{1,x,y\}$ using Algorithm[5] 8
end for
Lift the polynomials $f_{I,a}, \widehat{f}_{I,a,v}\ v \in \{1,x,y\}$ from their images modulo $\mu$ via the Chinese Remainder Algorithm
Compute $g_{I,a} = \gcd(f_{I,a}, \frac{\partial f_{I,a}}{\partial T})$
return $f_{I,a}, f_{I,a,v} = \frac{\widehat{f}_{I,a,v}}{g_{I,a}}\ v \in \{1,x,y\}$ the RUR-candidate of $\mathcal{T}$
```

with respect to $\mathcal{T}$ such that $\prod_{\mu\in\mathcal{P}} \mu > 2^m$. To derive the corresponding bit complexity, we need an upper bound on the number of unlucky prime numbers with respect to $\mathcal{T}$, that is, primes μ such that $\mathrm{Lc}_y(B)$ and A are not coprime when reduced modulo μ. A sufficient condition for $\phi_\mu(\mathrm{Lc}_y(B))$ and $\phi_\mu(A)$ to be coprime is that the integer $\mathrm{Lc}(\mathrm{Lc}_y(B))\,\mathrm{Lc}(A) Res_x(\mathrm{Lc}_y(B), A)$ does not vanish modulo μ [Yap 2000, §4.4]. Hence, the number of unlucky primes is bounded by the number of prime divisors of $\mathrm{Lc}(\mathrm{Lc}_y(B))\,\mathrm{Lc}(A) Res_x(\mathrm{Lc}_y(B), A)$. The number of prime divisors of an integer z is bounded by its bitsize. Indeed, its bitsize is $\lfloor \log z \rfloor + 1$ and its factorization into w (possibly identical) prime numbers directly yields that $2^w \leqslant \prod_{i=1}^{w} z_i = z = 2^{\log z} \leqslant 2^{\lfloor \log z \rfloor + 1}$. Using classical bounds on the bitsize of the resultant [Basu 2006, Proposition 8.46], an explicit upper bound U on the number of unlucky primes can be computed and the latter is in $\widetilde{O}(d_x(B)\tau')$ since $d_x(A) \leqslant d_x(B)$ by assumption.

To compute $\mathcal{P}$, we first compute $Res_x(\mathrm{Lc}_y(B)(x), A(x))$, then, we compute the first $m + U$ prime numbers and select among these primes those for which $\phi_\mu(\mathrm{Lc}(\mathrm{Lc}_y(B))\,\mathrm{Lc}(A) Res_x(\mathrm{Lc}_y(B), A)) \neq 0$. Since U is an upper bound on the number of unlucky primes, the number of the selected primes is then at least equal to m and their product is thus larger than 2^m. The bit complexity of computing $Res_x(\mathrm{Lc}_y(B), A)$ is in $\widetilde{O}_B(d_x(B)^2\tau')$ according to Theorem 2.4.17, that of computing the first $m + B$ prime numbers is in $\widetilde{O}_B(d_x(B)\tau' + m)$ and these primes are in $\widetilde{O}(d_x(B)\tau' + m)$ by [von zur Gathen 2003, Theorem 18.10].

The reduction of $\mathrm{Lc}(\mathrm{Lc}_y(B))\,\mathrm{Lc}(A)Res_x(\mathrm{Lc}_y(B), A)$ modulo all these primes can be computed via a remainder tree in a bit complexity that is soft linear in the total bitsize of the input by Theorem 2.5.3, which is dominated by the sum of the bitsize of the $\widetilde{O}(d_x(B)\tau' + m)$ primes, each of bitsize $\widetilde{O}(1)$. Hence computing $\mathcal{P}$ can be done using $\widetilde{O}_B(d_x(B)^2\tau' + m)$ bit operations.

In line 3, we compute the images of $\mathcal{T}$ modulo all the prime numbers in $\mathcal{P}$. This amounts to reduce $O(d_x(B)d_y(B))$ integer coefficients of bitsize $O(\tau')$ modulo at most m primes. One coefficient can be reduced with $\widetilde{O}_B(\tau' + m)$ bit operations using the same argument as above. Thus, all the coefficients are reduced using $\widetilde{O}_B(d_x(B)d_y(B)(\tau' + m))$ bit operations.

In line 5, we compute for the at most m primes in $\mathcal{P}$, the NRUR-candidate associated to $\phi_\mu(\mathcal{T})$. According to the proof of Lemma 5.3.4, for each prime number in $\mathcal{P}$ (of bitsize in $\widetilde{O}(1)$), this can be done by first computing the reduced lexicographic Gröbner basis of $\mathcal{T}$ which can be done with $\widetilde{O}_B(d_x(B)d_y(B))$ bit operations and then computing the NRUR-candidate using Algorithm 8, which needs $\widetilde{O}_B(d_{\mathcal{T}}^2)$ bit operations. Hence, the loop from line 4 to 6 needs $\widetilde{O}_B(m(d_x(B)d_y(B) + d_{\mathcal{T}}^2))$ bit operations.

The lifting step applies the Chinese Remainder Theorem for each of the $O(d_{\mathcal{T}})$ coefficients of $f_{I,a}, \widehat{f}_{I,a,v}$ $v \in \{1, x, y\}$. According to Theorem 2.5.4, this can be done with $\widetilde{O}_B(m)$ bit operations for one coefficient and thus, with $\widetilde{O}_B(md_{\mathcal{T}})$ for all the coefficients.

Finally, the RUR-candidate is deduced from the non-reduced one using one gcd computation and three Euclidean divisions between polynomials of degree at most $d_{\mathcal{T}}$ and bitsize in $O(m)$, which requires $\widetilde{O}_B(md_{\mathcal{T}}^2)$ bit operations according to Theorems 2.3.10 and 2.3.2. □

We consider now a regular triangular system $S_k = \{Sres_k(x, y), F_k(x)\}$ as returned by Algorithm 7 and denote by I_k the associated ideal. Applying the result of the previous lemma on S_k yields the following theorem. Recall that d and τ denote the maximum degree and bitsize of the input polynomials P and Q.

Theorem 5.4.5. *Let $S_k = \{Sres_k(x, y), F_k(x)\}$ with $d_y(Sres_k)d_x(F_k) = d_k$ and $T = x + ay$ be a linear form with a an integer of bitsize in $\widetilde{O}(1)$. Algorithm 9 computes the RUR-candidate of S_k associated to T using $\widetilde{O}_B((d^3 + d_k^2)(d^2 + d\tau) + d^5\tau)$ bit operations. Assuming that $Res_x(sres_k, F_k)$ is known, the previous bit complexity reduces to $\widetilde{O}_B((d^3 + d_k^2)(d^2 + d\tau))$.*

Proof. According to Theorem 2.4.16, $Sres_k(x, y)$ has degree in x in $O(d^2)$ and both $Sres_k(x, y)$ and $F_k(x)$ have bitsize in $d\tau$. In addition, since m denotes an upper bound on the bitsize of the coefficients of $f_{I_k,a}, \widehat{f}_{I_k,a,v}$, $v \in \{1, x, y\}$, then, by Lemma 5.3.2 and Remark 5.2.3, m is in $\widetilde{O}(d^2 + d\tau)$. The claim thus follows by applying the result of Lemma 5.4.4. □

Remark. Given a system $S = \{P, Q\}$ with $P, Q \in \mathbb{Z}[x, y]$ of total degree at most d and bitsize at most τ, we can use Algorithm 9 to design a first algorithm that computes the RUR-candidates of S. This algorithm proceeds first by computing the triangular decomposition of S over $\mathbb{Z}$ and then, for every regular triangular system S_k, it computes a RUR-candidate using Algorithm 9. The bit complexity of this algorithm is dominated by that of the triangular decomposition, which is in $\widetilde{O}_B(d^7 + d^6\tau)$ by Theorem 5.3.3. Indeed, according to Theorem 5.4.5 and noticing by Bézout's bound that $\sum_{k\in\{1,\dots,q\}} d_k \leqslant d^2$ (Lemma 5.3.2), the bit complexity for computing all the RUR-candidates from all the regular triangular systems is in $\widetilde{O}_B(d^6 + d^5\tau)$.

One advantage of this algorithm is that it prevents the possible coefficient swell that can result from the Gröbner basis computation, which highly speeds up the computation of the RUR-candidates from the triangular systems. As a consequence, the triangular decomposition step becomes the dominant part in the whole algorithm. However, one can also reduce the cost of this step by using multi-modular computations, as for the computation of the RUR-candidates. This is the topic of the next section.

5.4.3 RUR-candidates for $\mathbf{S = \{P, Q\}}$

We now extend the multi-modular approach used in Algorithm 9 to a general system $S = \{P, Q\}$. Next, we compare quantities defined when applying the triangular decomposition algorithm of Section 5.3 over $\mathbb{Z}$ and over $\mathbb{Z}_\mu$ for a given prime number μ. In particular, we consider for P and Q in $\mathbb{Z}[x, y]$, the polynomials F_i, G_i and the systems S_i. On the other hand, given P_μ and Q_μ in $\mathbb{Z}_\mu[x, y]$, the corresponding objects will be denoted with the superscript μ, that is by, F_i^μ, G_i^μ and S_i^μ.

The following result, which is analogous to that of Proposition 5.4.3, provides conditions on a prime number μ such that the NRUR-candidates computed for $S = \{P, Q\}$ using the algorithms of Section 5.3 are well specialized modulo μ. Recall that $Fres(P, Q)$ is the squarefree part of the resultant of P and Q, seen as polynomials in y from which are removed the common roots of their leading coefficients.

Definition 5.4.6. *Let $S = \{P, Q\}$ and $\{S_0, S_1, \dots, S_q\}$ be the regular triangular systems associated to S. A prime number μ is said to be lucky with respect to S if*

1. $\phi_\mu(\mathrm{Lc}_y(P)\ \mathrm{Lc}_y(Q)) \neq 0$,
2. *$Fres(P, Q)$ and $Fres(P_\mu, Q_\mu)$ have the same degree, and*
3. *$\forall k \in \{1, \dots, q\}$, μ is lucky with respect to the triangular system S_k (Definition 5.4.2).*

Proposition 5.4.7. *If* μ *is lucky with respect to* $S = \{P, Q\}$ *then,* $\forall k \in \{1, \dots, q\}$ $f_{I_k^\mu, \phi_\mu(a)} = \phi_\mu(f_{I_k,a})$ *and* $\widehat{f}_{I_k^\mu, \phi_\mu(a), v} = \phi_\mu(\widehat{f}_{I_k,a,v})$ *for* $v \in \{1, x, y\}$*.*

Proof. We first prove the following equality which is a consequence of the two first conditions of Definition 5.4.6, and of Lemma 5.2.4.

$$\phi_\mu(Fres(P, Q)) = Fres(P_\mu, Q_\mu). \tag{5.2}$$

$Fres(P_\mu, Q_\mu)$ divides $\overline{Res_y(P_\mu, Q_\mu)}$ which is equal to $\overline{\phi_\mu(Res_y(P, Q))}$ by the specialization property of subresultants since $\phi_\mu(\mathrm{Lc}_y(P)\ \mathrm{Lc}_y(Q)) \neq 0$. Moreover, according to Lemma 5.2.4, $\overline{\phi_\mu(Res_y(P, Q))}$ divides $\phi_\mu(\overline{Res_y(P, Q)})$. On the other hand, since we assume that $\mathrm{Lc}_y(P)$ and $\mathrm{Lc}_y(Q)$ are coprime, then $\phi_\mu(Fres(P, Q))$ is equal to $\phi_\mu(\overline{Res_y(P, Q)})$. Thus, $Fres(P_\mu, Q_\mu)$ divides $\phi_\mu(Fres(P, Q))$. Since by definition of luckiness, $Fres(P_\mu, Q_\mu)$ and $Fres(P, Q)$ have the same degree, it follows that $\phi_\mu(Fres(P, Q)) = Fres(P_\mu, Q_\mu)$.

Next, we prove that $\forall k \in \{1, \dots, q\}$ the systems $S_k = \{Sres_k, F_k\}$ are well specialized modulo μ or in other words that $\phi_\mu(Sres_k) = Sres_k^\mu$ and $\phi_\mu(F_k) = F_k^\mu$. The specialization of the NRUR-candidates then, follows directly from Proposition 5.4.3.

Since $\phi_\mu(\mathrm{Lc}_y(P)\,\mathrm{Lc}_y(Q)) \neq 0$, the specialization property of subresultants implies that $\forall k \in \{0, \dots, q\}$, $\phi_\mu(Sres_k) = Sres_k^\mu$. Next, we prove by induction that $\forall k \in \{1, \dots, q\}$, $\phi_\mu(F_k) = F_k^\mu$.

Initialization of the induction. According to the triangular decomposition algorithm, F_1^μ is the gcd-free part of $Fres(P_\mu, Q_\mu)$ with respect to $sres_1^\mu$ which is also that of $\phi_\mu(Fres(P, Q))$ with respect to $\phi_\mu(sres_1)$ by (5.2). According to Lemma 5.2.4, this gcd-free part divides $\phi_\mu(F_1)$ which is the image modulo μ of the gcd-free part of $Fres(P, Q)$ with respect to $sres_1$. Since μ is lucky with respect to S_1 i.e. $\phi_\mu(F_1)$ and $\phi_\mu(sres_1)$ are coprime, then $\phi_\mu(F_1)$ is the gcd-free part of $\phi_\mu(Fres(P, Q))$ with respect to $\phi_\mu(sres_1)$ and the property follows for $i = 1$.

Induction. Suppose that the property holds for all $i < k$, that is $\forall i < k$: $F_i^\mu = \phi_\mu(F_i)$. Then, as a consequence we have $G_k^\mu = \phi_\mu(G_k)$. Applying the same argument as above, it follows that $F_k^\mu = \phi_\mu(F_k)$. □

The following lemma shows that we can compute an explicit upper-bound on the number of unlucky primes with respect to $S = \{P, Q\}$. This bound will be used in Algorithm 10 to determine the probability of computing the correct RUR-candidates of the decomposition of S.

Lemma 5.4.8. *An upper bound on the number of unlucky primes with respect to* $S = \{P, Q\}$ *can be explicitly computed in terms of* d *and* τ*, and this bound is in* $\widetilde{O}(d^5 + d^4\tau)$*.*

Proof. To bound the number of unlucky prime numbers with respect to $S = \{P, Q\}$, we bound the number of prime numbers μ that violate at least one condition of Definition 5.4.6.

Number of primes μ such that $\phi_\mu(\mathrm{Lc}_y(P)\ \mathrm{Lc}_y(Q)) = 0$. For a prime number μ, $\phi_\mu(\mathrm{Lc}_y(P)\ \mathrm{Lc}_y(Q)) = 0$ if μ divides the gcd of all the coefficients of $\mathrm{Lc}_y(P)$ or the gcd of all the coefficients of $\mathrm{Lc}_y(Q)$. Similarly as in the proof of Theorem 5.4.5, the number of primes μ that divide one of the two previous gcds is bounded by the sum of their bitsize, which is at most equal to 2τ.

Number of primes μ such that $Fres(P, Q)$ and $Fres(P_\mu, Q_\mu)$ do not have the same degree. According to the definition of $Fres(P, Q)$, a prime number μ such that $\phi_\mu(\mathrm{Lc}_y(P))$ and $\phi_\mu(\mathrm{Lc}_y(Q))$ are coprime and the degree of the gcd of $Res_y(P, Q)$ and $\frac{Res_y(P,Q)}{\partial x}$ does not change after reduction modulo μ, implies that $Fres(P, Q))$ and $Fres(P_\mu, Q_\mu)$ have the same degree (Theorem 2.5.7). Hence, the prime numbers such that $Fres(P, Q)$ and $Fres(P_\mu, Q_\mu)$ do not have the same degree, are bounded by the sum of the number of primes for which $\phi_\mu(\mathrm{Lc}_y(P))$ and $\phi_\mu(\mathrm{Lc}_y(Q))$ are not coprime and the number of those for which the degree of the gcd of $Res_y(P, Q)$ and $\frac{Res_y(P,Q)}{\partial x}$ changes after reduction modulo μ. According to [Yap 2000, Lemma 4.12], given two univariate polynomials in $\mathbb{Z}[x]$ of degree at most d' and bitsize at most τ', the product of all μ, such that the degree of the gcd of the two polynomials changes when the polynomials are considered modulo μ, is bounded by $(2^{\tau'}\sqrt{d'+1})^{2d'+2}$. The number of such primes μ is bounded by the bitsize of this bound, and thus is bounded by $(d'+1)\,(2\tau' + \log(d'+1)) + 1$. For $\mathrm{Lc}_y(P)$ and $\mathrm{Lc}_y(Q)$, $d' \leqslant d$ and $\tau' \leqslant \tau$ which gives a first bound in $\widetilde{O}(d\tau)$, while for $Res_y(P, Q)$ and $\frac{Res_y(P,Q)}{\partial x}$, $d' \leqslant 2d^2$ and τ' is in $\widetilde{O}(d^2 + d\tau)$ ([Basu 2006, Proposition 8.46]) which gives a second bound in $\widetilde{O}(d^4 + d^3\tau)$. Summing these two bounds, yields a bound in $\widetilde{O}(d^4 + d^3\tau)$ on the the number of prime μ such that $Fres(P, Q)$ and $Fres(P_\mu, Q_\mu))$ do not have the same degree.

Number of primes μ that are unlucky with respect to one of the S_k with $k \in \{1, \ldots, q\}$. As shown in the proof of Theorem 5.4.5, the number of prime numbers that are unlucky with respect to a triangular system S_k is bounded by a quantity in $\widetilde{O}(d^4 + d^3\tau)$. Since the decomposition of $S = \{P, Q\}$ yields at most d such triangular systems, the number of primes that are unlucky with respect to one of the S_k with $k \in \{1, \ldots, q\}$ is thus in $\widetilde{O}(d^5 + d^4\tau)$.

The result follows by summing the three previous bounds. □

Although the prime numbers in Algorithm 10 are chosen arbitrary without prior verification, it is still possible during the algorithm, to detect (and hence to discard) some sequences of NRUR-candidates that correspond to unlucky primes just by looking to some intermediate degrees. The following lemma which is a slight variation on Proposition 5.4.7 gives characterizations in terms of some intermediate degrees, of unlucky prime numbers with respect to S. These characterizations are useful to discard during the computation, all sequences of NRUR-candidates that correspond to unlucky prime numbers, once a lucky prime has been chosen, which allows to improve the probability of success of Algorithm 10. Another important result stated in this lemma, is that the number of solutions (counted with multiplicities) of the RUR-candidates computed by Algorithm 10 is greater or equal than the number of solutions of the input system $S = \{P, Q\}$. This result is used in Section 5.4.4.1 to check the correctness of the RUR-candidate output by Algorithm 10 by simply verifying that all the candidate solutions are indeed solutions of system.

Lemma 5.4.9. *Let* $S = \{P, Q\}$ *and* $\{S_0, S_1, \ldots, S_q\}$ *be the regular triangular systems associated to* S *with* $S_i = \{F_i(x), Sres_i(x, y)\}$ *as described in Section 5.3.1. Let* μ *be a prime number such that* $\phi_\mu(\mathrm{Lc}_y(P)\ \mathrm{Lc}_y(Q)) \neq 0$. *Then,*

1. *the degree of* $Fres(P, Q)$ *is at least that of* $Fres(P_\mu, Q_\mu)$.

If $Fres(P, Q)$ *and* $Fres(P_\mu, Q_\mu)$ *have the same degree, then,*

2. $\sum_{i=1}^{q} i\ \times \deg(F_i) \leqslant \sum_{i=1}^{q} i \times \deg(F_i^\mu)$ *and the inequality is strict if and only if* μ *is* **unlucky** *with respect to one of the* S_k *with* $k \in \{1, \ldots, q\}$.

Proof. The first point is a direct consequence of the proof of Proposition 5.4.7. For the second point, we first prove, by induction, that for $i = 0, \ldots, q$, $\phi_\mu(G_i)$ divides G_i^μ.

Initialization of the induction. It follows from the proof of Proposition 5.4.7 that $\phi_\mu(G_0) = \phi_\mu(Fres) = Fres(\phi_\mu) = G_0^\mu$.
Induction. Suppose that the property holds for $k-1$, that is: $\phi_\mu(G_{k-1})$ divides G_{k-1}^μ. Since $G_k = \gcd(G_{k-1}, sres_k)$, this implies that $\phi_\mu(G_k)$ divides $\gcd(\phi_\mu(G_{k-1}), \phi_\mu(sres_k))$. By the specialization of subresultants, we have that $\phi_\mu(G_k)$ divides $\gcd(\phi_\mu(G_{k-1}), sres_k(\phi_\mu))$. Using the induction hypothesis we then conclude that $\phi_\mu(G_k)$ divides $\gcd(G_{k-1}^\mu,\ sres_k(\phi_\mu))$, which is equal to G_k^μ, hence the property holds for k.

We have $Fres = \prod_{i=1}^{q} F_i$ and $deg(Fres) = deg(\phi_\mu(Fres))$, which implies that $\forall j \in \{1, \ldots, q\}$, $deg(F_j) = deg(\phi_\mu(F_j))$. Since $G_j = \prod_{i>j}^{q} F_i$, then for every $j \in \{1, \ldots, q\}$, $deg(G_j) = deg(\phi_\mu(G_j))$. Together with the previous fact that $\phi_\mu(G_i)$ divides G_i^μ, we conclude that $deg(G_j) \leqslant deg(G_j^\mu)$.

Denote by n_i (resp. m_i) the degree of F_i (resp. F_i^μ). Then we have $deg(G_j) = \sum_{i>j}^{q} n_i$ and $deg(G_i^\mu) = \sum_{i>j}^{q} m_i$ and the inequality $deg(G_j) \leqslant$

$deg(G_j^\mu)$ rewrites as $\sum_{i=j}^{q} n_i \leqslant \sum_{i=j}^{q} m_i$ for every $j \in \{1, \ldots, q\}$. Summing these inequalities for j from 1 to q yields $\sum_{i=1}^{q}(i \times n_i) \leqslant \sum_{i=1}^{q}(i \times m_i)$.

Suppose now that μ is unlucky with respect to some S_k with $k \in \{1, \ldots, q\}$. In that case, $\phi_\mu(G_k)$ strictly divides G_k^μ and $deg(\phi_\mu(G_k)) < deg(G_k^\mu)$. Applying the same argument as above we obtain that $\sum_{i=1}^{q}(i \times n_i) < \sum_{i=1}^{q}(i \times m_i)$. □

Algorithm 10 RUR-candidates for $S = \{P, Q\}$ (**Monte-Carlo algorithm**)

```
Require: $P, Q$ in $\mathbb{Z}[x, y]$ defining a finite set of solutions in $\mathbb{C}^2$ with Lc($P$) and Lc($Q$) coprime and a linear form $T = x + ay$
Ensure: $\mathcal{R}$ the set of the RUR-candidates of the decomposition of $S = \{P, Q\}$, associated to $T$.
Compute an integer $m$ such that $2^m$ is greater than the maximum size of the coefficients of $f_{I,a}, \widehat{f}_{I,a,v},\ v \in \{1, x, y\}$ (see Remark 5.2.3)
Compute an upper bound $B$ on the number of unlucky primes with respect to $S = \{P, Q\}$ (see Lemma 5.4.8)
Compute a set $U$ of prime numbers $\mu > d^2$ whose cardinal is $2B$
$A_{\mathcal{R}} = \emptyset$
repeat
    Choose uniformly at random a set $\mathcal{P}$ of $2m$ primes $\mu$ in $U$ such that $\phi_\mu(\mathrm{Lc}(P)\,\mathrm{Lc}(Q)) \neq 0$
    Compute $S_\mu = \{P_\mu, Q_\mu\}$ for all $\mu$ in $\mathcal{P}$
    for $\mu \in \mathcal{P}$ do
        Compute the sequence $\mathcal{R}^\mu$ of the NRUR-candidates of $S_\mu = \{P_\mu, Q_\mu\}$ using the algorithm of Section 5.3
        $A_{\mathcal{R}} = A_{\mathcal{R}} \cup \mathcal{R}^\mu$
    end for
    Discard from $A_{\mathcal{R}}$ every sequence $\mathcal{R}^\mu$ of NRUR-candidates such that the degree of $Fres^\mu$ is not maximal
    Discard from $A_{\mathcal{R}}$ every sequence $\mathcal{R}^\mu$ of NRUR-candidates such that the sum of the degrees of $S_k^\mu$ is not minimal[6]
until $A_{\mathcal{R}}$ contains at least $m$ sequences of NRUR-candidates
Lift, with the Chinese Remainder Algorithm, the sequences of NRUR-candidates in $A_{\mathcal{R}}$ to a sequence $\mathcal{R}$
Compute for every NRUR-candidate in $\mathcal{R}$ the corresponding RUR-candidate (Definition 5.2.2)
return $\mathcal{R}$
```

---

[6] We call *degree* of a regular triangular system $\mathcal{T} = \{B(x, y), A(x)\}$ the product of the degree of B in y and the degree of A which is also the number of complex solutions of $\mathcal{T}$ counted with multiplicities.

Theorem 5.4.10. *Let $S = \{P, Q\}$ with P, Q of total degree at most d and maximum bitsize τ such that $Lc(P)$ and $Lc_y(Q)$ are coprime, and $T = x + ay$ be a linear form with integer a of bitsize in $\widetilde{O}(1)$. Let m be an integer in $\widetilde{O}_B(d^2 + d\tau)$ that is larger than the maximum bitsize of the coefficients of the RUR-candidates of S (see Remark 5.2.3). Algorithm 10 computes the RUR-candidates of the decomposition of S, associated to T using an expected number of $\widetilde{O}_B(d^6 + d^5\tau)$ bit operations with a probability of success larger than $1 - \frac{1}{2^{2m}}$.*

Proof. We first bound the expected number of operations to compute the RUR-candidates and then estimate the probability that the computed RUR-candidates are correct.

Expected number of bit operations. In Line 3, we compute a set U of $2B$ prime numbers larger than d^2 such that $2B$ is in $\widetilde{O}(d^5 + d^4\tau)$. For computing U, we compute the first $2B + d^2$ prime numbers and reject those that are smaller than d^2. The bit complexity of computing the r first prime numbers is in $\widetilde{O}_B(r)$ and their maximum is in $\widetilde{O}(r)$ [von zur Gathen 2003, Thm. 18.10]. We can thus compute the set of primes U with $\widetilde{O}_B(d^5 + d^4\tau)$ bit operations and these primes are in $\widetilde{O}(d^5 + d^4\tau)$.

In the outer loop (Line 5 to 14), we successively select sets $\mathcal{P}$ of random prime numbers in U and compute for every prime μ therein, the corresponding NRUR-candidates using the algorithms of Section 5.3 (see Theorem 5.3.6) over $\mathbb{Z}_\mu$. We stop when the condition in Line 14 is satisfied. To estimate the corresponding expected number of bit operations, we analyse the cost of one iteration and then determine the expected number of iterations.

In Line 6, we select among the prime numbers in U, a set $\mathcal{P}$ of $2m$ primes that do not cancel $\mathrm{Lc}_y(P)\,\mathrm{Lc}_y(Q)$. As shown in the proof of Proposition 5.4.8 there is at most 2τ prime number μ such that $\phi_\mu(\mathrm{Lc}_y(P)\,\mathrm{Lc}_y(Q)) = 0$, hence, to compute $\mathcal{P}$, we select $2m + 2\tau = \widetilde{O}(d^2 + d\tau)$ prime numbers from U and reject those which identically cancel $\mathrm{Lc}_y(P)\,\mathrm{Lc}_y(Q)$. This amounts to reduce $\mathrm{Lc}_y(P)\,\mathrm{Lc}_y(Q)$ modulo $\widetilde{O}(d^2 + d\tau)$ prime numbers, which can be done via remainder tree in $\widetilde{O}(d^3 + d^2\tau)$. We then compute for all the prime μ in $\mathcal{P}$ the reduction of P and Q modulo μ which can be done using the same strategy with $\widetilde{O}(d^4 + d^3\tau)$ bit operations. In Line 8, for every prime number μ of $\mathcal{P}$, we call the algorithms of Section 5.3 (see Theorem 5.3.6) to compute the NRUR-candidates of $\{P_\mu, Q_\mu\}$. According to Theorems 5.3.3 and 5.3.5, the arithmetic complexity of the whole algorithm is in $\widetilde{O}(d^4)$. On the other hand μ has bitsize in $O(\log(d^5 + d^4\tau))$, thus, the bit complexity of one call is in $\widetilde{O}_B(d^4)$ which yields $\widetilde{O}_B(d^6 + d^5\tau)$ for computing all the sequences R^μ. Finally, the bit complexities of Line 12 and 13 can be neglected. Hence, the bit complexity of one iteration of the outer loop is in $\widetilde{O}_B(d^6 + d^5\tau)$.

Choosing randomly a prime number μ in U, the probability that μ is lucky with respect to S is larger than $\frac{1}{2}$. Thus choosing randomly $2m$ primes numbers in U, the probability that half of them are lucky with respect to S is larger than $\frac{1}{2}$. Hence, in average, at most two iterations of the outer loop suffices to obtain at least m lucky prime numbers with respect to S.

Once at least m lucky prime numbers have been chosen, the halting condition in Line 14 is necessarily satisfied. Indeed, according to Lemma 5.4.9, none of the corresponding sequences of NRUR-candidates in $A_{\mathcal{R}}$ are discarded in Lines 12 (resp. 13) and we end up with a set $A_{\mathcal{R}}$ containing at least m sequence of NRUR-candidates. The expected number of bit operations of the outer loop is then in $\widetilde{O}_B(d^6 + d^5\tau)$. Finally, the bit complexities of Lines 15 and 16 can be deduced from the proof of Theorem 5.4.5 and are respectively equal to $\widetilde{O}_B(d^4 + d^3\tau)$ and $\widetilde{O}_B(d^6 + d^5\tau)$.

Probability of success. Algorithm 10 fails to output RUR-candidates of the input system, if at Line 14, the set $A_{\mathcal{R}}$ contains at least m sequences of NRUR-candidates and none of them correspond to a prime μ that is lucky with respect to S. Indeed if $A_{\mathcal{R}}$ contains at least one sequence of NRUR-candidates that correspond to such a prime number, then according to Lemma 5.4.9, all the other sequences that correspond to unlucky prime numbers with respect to S will be discarded in Line 12 or 13. Hence, the algorithm finally finds m sequences corresponding to lucky primes and succeeds, according to Proposition 5.4.7, to return the RUR-candidates. Hence, the probability that Algorithm 10 fails is at most the probability that the first $2m$ prime numbers are all unlucky with respect to S. Since a randomly chosen prime in U is unlucky with probability less than $\frac{1}{2}$, the $2m$ randomly chosen prime numbers are unlucky with probability at most $\frac{1}{2^{2m}}$. It follows that Algorithm 10 succeeds with probability larger than $1 - \frac{1}{2^{2m}}$. $\square$

5.4.4 Las-Vegas algorithms

We present in the following, two methods for checking that the RUR-candidates of S computed by Algorithm 10 are indeed RURs. To do that, we need to check that the RUR-candidates computed by Algorithm 10 are correct, but also that the chosen linear form is separating. Applying these methods to the algorithm above, that is, running it iteratively until the check is satisfied yields naturally two Las-Vegas algorithms for computing the RURs of $S = \{P, Q\}$ whose expected bit complexities are respectively in $\widetilde{O}_B(d^7 + d^6\tau)$ and $\widetilde{O}_B(d^6 + d^5\tau)$.

More precisely, these Las-Vegas algorithms choose randomly a linear form $x + ay$ with a an integer between 0 and $2d^4$ and run Algorithm 10 for this form. This process is repeated until the check is positive. Since the number of non-separating forms is less than d^4, the chosen linear form has probability larger than $\frac{1}{2}$ to be separating, and as we have seen in the proof of Theorem 5.4.10, at most two iterations of the outer loop suffices, in average, to obtain at least m lucky prime numbers with respect to S; hence Algorithm 10 runs at most four times on average before the check is positive.

Note that, the first Las-Vegas algorithm, of complexity $\widetilde{O}_B(d^7 + d^6\tau)$, has been implemented and its practical efficiency is demonstrated in Chapter 7. The second algorithm, which has better complexity $\widetilde{O}_B(d^6 + d^5\tau)$, is more recent and has not yet been implemented. We present these two approaches in the two following subsections.

5.4.4.1 $\widetilde{\mathbf{O}}_{\mathbf{B}}(\mathbf{d}^7 + \mathbf{d}^6\tau)$-expected-time Las-Vegas algorithm

Lemma 5.4.9 shows that for prime numbers μ such that $\phi_\mu(\mathrm{Lc}_y(P)\ \mathrm{Lc}_y(Q)) \neq 0$ and $Fres(P, Q)$ and $Fres(P_\mu, Q_\mu)$ have the same degree, if the solution set computed by Algorithm 10 is not correct (i.e. it is not equal to the solution set of $\{P, Q\}$), then it cannot be a strict subset of the solutions of P and Q counted with multiplicities. More precisely, some root of the lifted RUR-candidates is not solution of the input system or some multiplicity is too large.

This suggests a method to certify Algorithm 10 that consists in computing RUR-candidates only modulo prime numbers satisfying the two previous conditions and to add at the end of the algorithm, a verification step that checks whether the roots of the obtained RUR-candidates are indeed roots of the original system $S = \{P, Q\}$ with their multiplicities in the RUR-candidates smaller or equal to their multiplicities in their fibers with respect to S. Note that this verification checks at the same time that the RUR-candidates computed by Algorithm 10 are corrects and also that the chosen linear form for computing these RUR-candidates is separating.

Checking whether the roots of a RUR-candidate $\{f_{I,a}, f_{I,a,1}, f_{I,a,x}, f_{I,a,y}\}$ are also roots of S can be done by substituting the coordinates $x = \frac{f_{I,a,x}(T)}{f_{I,a,1}(T)}$ and $y = \frac{f_{I,a,y}(T)}{f_{I,a,1}(T)}$ in P and Q and checking that $f_{I,a}$ divides their numerators. Checking, for each solution of a RUR-candidate, that its multiplicity is smaller or equal to that in its fiber in S can be done in a similar way using derivatives with respect to y of P and Q. More precisely, if $\mathbf{p} = (\alpha, \beta)$ is a solution with multiplicity m in a RUR-candidate, one has to show that the derivatives $\frac{\partial P(\alpha,y)}{\partial y}, \ldots, \frac{\partial^{m-1} P(\alpha,y)}{\partial^{m-1} y}$ and $\frac{\partial Q(\alpha,y)}{\partial y}, \ldots, \frac{\partial^{m-1} Q(\alpha,y)}{\partial^{m-1} y}$ all vanish at β. An effective way to do so is to consider the decomposition of the RUR-candidates output by Algorithm 10 according to the multiplicity of their solutions. More precisely, each RUR-candidate $R_k = \{f_{I_k,a}, f_{I_k,a,1}, f_{I_k,a,x}, f_{I_k,a,y}\}$ is decomposed into a set of single-multiplicity RUR-candidates $R_{k,i} = \{f_{I_k,a,i}, f_{I_k,a,1,i}, f_{I_k,a,x,i}, f_{I_k,a,y,i}\}$

such that $f_{I_k,a} = \Pi_i f^i_{I_k,a,i}$. We then verify for i increasing from 2 that the solutions of the RUR-candidate of multiplicity i are also solutions of the $(i-1)^{\text{th}}$ derivatives of P and Q with respect to y as described above. Before giving the bit complexity of the verification, we first analyze the bit complexity of decomposing the RUR-candidates output by Algorithm 10 into RUR-candidates whose solutions have the same multiplicity.

Lemma 5.4.11. *Let $R_k = \{f_{I_k,a}, f_{I_k,a,1}, f_{I_k,a,x}, f_{I_k,a,y}\}$ for $k = 1, \ldots, q$ be the RUR-candidates output by Algorithm 10. Computing the decomposition of all the R_k into single-multiplicity RUR-candidates can be done using $\widetilde{O}_B(d^7 + d^6\tau)$ bit operations.*

Proof. The decomposition of $R_k = \{f_{I_k,a}, f_{I_k,a,1}, f_{I_k,a,x}, f_{I_k,a,y}\}$ is obtained by first computing the squarefree decomposition $f_{I_k,a} = \Pi_i f^i_{I_k,a,i}$ and second, by computing for each factor $f_{I_k,a,i}$ the corresponding polynomials $f_{I_k,a,v,i} = f_{I_k,a,v} \bmod f_{I_k,a,i}$ for $v \in \{1, X, Y\}$ (see [Rouillier 1999, §5.3] for details). Computing the squarefree decomposition of $f_{I_k,a}$ can be done with $\widetilde{O}_B(dd_k^2\tau_k)$ bit operations using Yun's squarefree decomposition algorithm [von zur Gathen 2003, Algorithm 14.21] together with Theorem 2.3.10, while computing the polynomials $\{f_{I_k,a,1,i}, f_{I_k,a,x,i}, f_{I_k,a,y,i}\}$ for each multiplicity i costs $\widetilde{O}_B(d_k^2\tau_k)$ bit operations according to Theorem 2.3.2. Since there is at most d multiplicities, the decomposition of R_k into single-multiplicity RUR-candidates can be done using $\widetilde{O}_B(dd_k^2\tau_k)$. Summing over all the R_k for $k = 1, \ldots, d$ and using the fact that $\tau_k = \widetilde{O}(d^2 + d\tau)$ and that $\sum_k d_k \leqslant d^2$ by Lemma 5.3.2, Proposition 4.3.1 and Bézout's bound, we obtain an overall complexity of $\widetilde{O}_B(d^7 + d^6\tau)$ bit operations. □

Lemma 5.4.12. *Checking that the solutions of all the RUR-candidates $R_{k,i}$ are solutions of $\{P, Q\}$ with the right multiplicities in fiber can be done using $\widetilde{O}_B(d^7 + d^6\tau)$ bit operations.*

Proof. Let $R_{k,i} = \{f_{I_k,a,i}, f_{I_k,a,1,i}, f_{I_k,a,x,i}, f_{I_k,a,y,i}\}$ be the RUR-candidate of multiplicity i resulting from the decomposition of the the RUR-candidate $R_k = \{f_{I_k,a}, f_{I_k,a,1}, f_{I_k,a,x}, f_{I_k,a,y}\}$, and denote by $d_{k,i}$ and $\tau_{k,i}$ the maximum degree and bitsize of its polynomials. Checking that the solutions of $R_{k,i}$ are solutions of P (resp. Q) is done by first, computing the numerator of $P(\frac{f_{I_k,a,x,i}}{f_{I_k,a,1,i}}, \frac{f_{I_k,a,y,i}}{f_{I_k,a,1,i}})$ and then by checking that $f_{I_k,a,i}$ divides this numerator. We show in Lemma 6.2.1 that the computation of the numerator of $P(\frac{f_{I_k,a,x,i}}{f_{I_k,a,1,i}}, \frac{f_{I_k,a,y,i}}{f_{I_k,a,1,i}})$ can be done in $\widetilde{O}_B(d^3 d_{k,i}\tau_{k,i})$ and yields a polynomial of degree in $O(dd_{k,i})$. Testing the divisibility of this numerator by $f_{I_k,a,i}$ is achieved using $\widetilde{O}_B(dd_{k,i}(d_{k,i} + \tau_{k,i}))$ bit operations according to Theorem 2.3.3. Hence, checking that the solutions of $R_{k,i}$ are solutions of $\{P, Q\}$ can be done using

$\widetilde{O}_B(d^3 d_{k,i}\tau_{k,i} + dd_{k,i}^2)$ bit operations. On the other hand, since $R_{k,i}$ corresponds to solutions of multiplicity i, checking that these solutions have the right multiplicity in their fibers amounts to repeat i times the above computation with the successive derivatives of P and Q which yields a bit complexity in $\widetilde{O}_B(i \times (d^3 d_{k,i}\tau_{k,i} + dd_{k,i}^2))$ for each RUR-candidate $R_{k,i}$. Now, summing over all the RUR-candidates and using the fact that $\tau_{k,i} = \widetilde{O}(d^2 + d\tau)$ and that $\sum id_{k,i} \leqslant d^2$ by Lemma 5.3.2, Proposition 4.3.1 and Bézout's bound, we obtain an overall bit complexity in $\widetilde{O}_B(d^7 + d^6\tau)$. $\square$

The following theorem gives the expected bit complexity of computing the RURs of S by repeating Algorithm 10 until the previous verification is positive. This follows from Theorem 5.4.10 and Lemmas 5.4.11 and 5.4.12 since, as mentioned at the beginning of Section 5.4.4, Algorithm 10 runs at most four times on average before the check is positive.

Theorem 5.4.13. *Let $S = \{P, Q\}$ with P, Q of total degree at most d and maximum bitsize τ. The RURs of the decomposition of S and an associated separating linear form $T = x + ay$ with integer a of bitsize in $\widetilde{O}(1)$, can be computed using an expected number of $\widetilde{O}_B(d^7 + d^6\tau)$ bit operations.*

5.4.4.2 $\mathbf{\widetilde{O}_B(d^6 + d^5\tau)}$-expected-time Las-Vegas algorithm

As shown in the proof of Proposition 5.4.7, a sufficient condition for the RUR-candidates computed by Algorithm 10 to be correct is that, the systems $S_k = \{Sres_k, F_k\}$ are well-specialized modulo the selected prime numbers. This means that for each prime number μ used to lift the RUR-candidates, the equalities $\phi_\mu(Sres_k) = Sres_k^\mu$ and $\phi_\mu(F_k) = F_k^\mu$ $\forall k \in \{1, \dots, q\}$, hold. An effective way to check that this is indeed the case, is to lift over $\mathbb{Z}$ the polynomials $sres_k$ and G_k for $k = 1, \dots, q$ (see Algorithm 6), modulo the primes μ used for lifting the RUR-candidates, and to check that, for every k, G_k divides both $sres_k$ and G_{k-1}.[7] If so, then the systems S_k are well-specialized and so that for the RUR-candidates, which implies that the RUR-candidates lifted over $\mathbb{Z}$ are corrects.

According to Theorem 2.5.4, lifting the polynomials $sres_k$ and G_k for $k = 1, \dots, q$ over $\mathbb{Z}$ can be done using $\widetilde{O}_B(d^5 + d^4\tau)$ bit operations, while testing for the divisibility of $sres_k$ by G_k for $k = 1, \dots, q$, can also be done using $\widetilde{O}_B(d^5 + d^4\tau)$ bit operations according to Theorem 2.3.3.

---

[7] It is assumed here that the product of the primes used to lift the polynomials $sres_k$ and G_k for $k = 1, \dots, q$ is larger than their maximum coefficient. However, this assumption is always fulfilled since we require in Algorithm 10 this product to be larger than an upper bound on the maximum coefficient of the RUR-candidates which, in turn, also bounds the maximum coefficient of $sres_k$ and G_k.

In a second step, we need to check that the chosen linear form $X + aY$ is separating, i.e. that the RUR-candidates of S computed by Algorithm 10 are actually RURs. To do that, we use the same strategy as in [Niang Diatta 2008] where the genericity of a curve f is tested by first computing a triangular decomposition of the system $\{f, \frac{f}{\partial f}\}$ and then by checking algebraically that for each triangular system $\{Sres_k, F_k\}$, and for each root α of F_k the polynomial $Sres_k(\alpha, Y)$ admits only one root β. In our case, in order to check that the linear form $X + aY$ is separating, we consider the system $S_a = \{P(T - aY, Y), Q(T - aY, Y)\}$, on which we apply the previous test. More precisely, we first compute the triangular decomposition of S_a and then, for each triangular system $\{Sres_k(T, Y), F_k(T)\}$ we check that for each root γ of F_k the polynomial $Sres_k(\gamma, Y)$ admits only one root β.

From the complexity point of view, the triangular decomposition of S_a can be computed using an expected bit complexity in $\widetilde{O}_B(d^6 + d^5\tau)$. Indeed, according to Theorem 2.4.17, computing the polynomials $Sres_k(T, Y)$ for $k = 1, \ldots, q$ can be done using $\widetilde{O}_B(d^6 + d^5\tau)$ bit operations while matching the latter with the corresponding factor of the resultant (see Algorithm 6) can be done similarly as above using an expected number of $\widetilde{O}_B(d^5 + d^4\tau)$ bit operations (see Theorem 2.5.8). Finally, checking that for each triangular system $\{Sres_k(T, Y), F_k(T)\}$, $\forall\gamma$ root of F_k the polynomial $Sres_k(\gamma, Y)$ admits only one root β can be done in $\widetilde{O}_B(d^6 + d^5\tau)$ according to Theorem 2.3.3 (see [Diatta 2009, Theorem 3.3.9]).[8]

Similarly as in the previous section, we obtain the following result.

Theorem 5.4.14. *Let $S = \{P, Q\}$ with P, Q of total degree at most d and maximum bitsize τ. The RURs of the decomposition of S and an associated separating linear form $T = x + ay$ with integer a of bitsize in $\widetilde{O}(1)$, can be computed using an expected number of $\widetilde{O}_B(d^6 + d^5\tau)$ bit operations.*

5.5 Conclusion

We focused in this chapter on the practical efficiency of computing Rational Univariate Representations of bivariate systems. We present an approach that combines two ingredients. A decomposition of the initial system into several RURs using a triangular decomposition based on subresultants followed by several RUR computations, and the use of a multi-modular-based approach that prevents intermediate coefficients growth.

---

[8] Note that this complexity improves by a factor d the one stated in [Diatta 2009, Theorem 3.3.9]. This is due to the use of Theorem 2.3.3 which claims that testing the divisibility can be done $O(d)$ times faster than performing the Euclidean division.

To overcome the correctness problems induced by the multi-modular approach, we propose two methods for checking that the computed RURs correctly encode the solutions of the input system. This yields two Las-Vegas algorithms with expected bit complexity respectively in $\widetilde{O}_B(d^7 + d^6\tau)$ and $\widetilde{O}_B(d^6 + d^5\tau)$.

In the global process of solving bivariate systems via Rational Univariate Representations, this brings the complexity of the symbolic part (the computation of the Rational Univariate representation), in Las-Vegas setting, to that of the numerical isolation of the solutions (see Chapter 6).

In addition of the speedup induced by the multi-modular approach, another notable advantage of this approach is that it splits the initial system $S = \{P, Q\}$ into several smaller systems, which are naturally easier to handle. This is particularly the case when it comes to computing with the solutions of a bivariate system, where it is preferable to work with these solutions represented by a set of "small" RURs instead of one single RUR.

From a practical point of view, we implemented the Las-Vegas algorithm described in Section 5.4.4.1 (Theorem 5.4.13). This algorithm turns out to be very efficient as demonstrated in Chapter 7. Furthermore, the multi-modular approach naturally yields a multi-thread implementation which is even more efficient.

Finally, as mentioned in the introduction, the second verification method has not yet been implemented. Although of better complexity, it is not clear at all that the latter will outperform in practice the current implementation. This is mainly due to the fact that checking whether the chosen linear form is separating requires to perform a second triangular decomposition with the sheared system $S_a = \{P(t - ay, y), Q(t - ay, y)\}$ (see Section 5.4.4.2), which will at least double the running time of the algorithm. Currently, we are investigating an alternative method that runs in the same complexity that is $\widetilde{O}_B(d^6 + d^5\tau)$, for checking that the chosen linear form is separating using only the RUR-candidates computed by Algorithm 10.

Chapter 6

Implications and applications

Contents

We present in this chapter some implications and applications enlightening the advantages of computing a RUR of a system. We start with the problem of isolating the solutions of a bivariate system, that is, computing boxes with rational coordinates that isolate the solutions. As mentioned along this thesis, this is an important problem which is actually an integral part of the problem of solving an algebraic system of equations. We show in Section 6.1 that the complexity of computing isolating boxes from a RUR does not exceed the complexity of computing the RUR itself even when the RUR is computed in a probabilistic setting (as in Chapter 5). Hence adding the isolation step to the algorithms described in the previous chapters does not change their bit complexity.

We also address in Sections 6.2 and 6.3 two important applications of the RUR, that is, the evaluation of the sign of a bivariate polynomial at a real solution of the system and the computation of rational parametrizations of systems defined by several equality and inequality constraints.

The last section of this chapter, is devoted to the problem of computing the topology of curves. We show in particular that the application of our new solver (consisting of the Las-Vegas algorithm of Chapter 5 and the isolation algorithm of Section 6.1) to the algorithm of [Cheng 2010] yields several algorithmic and complexity improvements.

Remark. In the three following sections, we first assume for simplicity that we work with an arbitrary bivariate ideal I whose solutions are given by a

RUR $\{f_{I,a}, f_{I,a,1}, f_{I,a,X}, f_{I,a,Y}\}$ with polynomials of degree bounded by d_r and integer coefficients of bitsize bounded by τ_r. The obtained results are applied afterward for the RURs of the ideal $\langle P, Q\rangle$, computed in Chapter 4 or 5 (using the bitsize bound given by Proposition 4.3.1), in order to obtain complexities in terms of the degree and the bitsize of the input polynomials P and Q.[1]

6.1 Isolating boxes from a RUR

We consider here the problem of isolating boxes for the solutions of a system $\{P, Q\}$, given its RUR. We presented in [Bouzidi 2013b] an algorithm of bit complexity in $\widetilde{O}_B(d^8 + d^7\tau)$ for that problem. We present here an improved version of this algorithm whose complexity is in $\widetilde{O}_B(d^6 + d^5\tau)$.

Given a RUR $\{f_{I,a}, f_{I,a,1}, f_{I,a,X}, f_{I,a,Y}\}$, isolating boxes for the real solutions can be obtained by first computing isolating intervals for the real roots of the univariate polynomial $f_{I,a}$, and then evaluating the rational fractions $\frac{f_{I,a,X}}{f_{I,a,1}}$ and $\frac{f_{I,a,Y}}{f_{I,a,1}}$ by interval arithmetic. When these isolating intervals are sufficiently refined, the computed boxes are necessarily disjoint and thus isolating. The following theorem analyzes the bit complexity of this algorithm.

Theorem 6.1.1. *Let $\{f_{I,a}, f_{I,a,1}, f_{I,a,X}, f_{I,a,Y}\}$ be the RUR of an ideal I and let d_r and τ_r be upper bounds for the degree and the bitsize of its polynomials. Isolating boxes for the solutions of I can be computed in $\widetilde{O}_B(d_r^3 + d_r^2\tau_r)$ bit operations. The vertices of these boxes have bitsize in $\widetilde{O}(d_r^2 + d_r\tau_r)$.*

Proof. Let $f_{I,a,t} = f_{I,a,X} + af_{I,a,Y}$. For every real solution σ of I, let $J_{X,\sigma} \times J_{Y,\sigma}$ be a box containing it. Instead of checking the overlap of boxes in two dimensions, projecting these boxes via the separating element $X + aY$ enables a simpler intersection test on the real line. Indeed, a sufficient condition for these boxes to be isolating is that the intervals $\{J_{X,\sigma} + aJ_{Y,\sigma}\}_{\sigma\in V(I)}$ do not overlap. This leads to the following algorithm.

First we compute isolating intervals J_γ for the roots of $f_{I,a}$ (we assume a dyadic representation of the form $[\frac{a}{2^b}, \frac{a+1}{2^b}]$, $a \in \mathbb{Z}$, $b \in \mathbb{N}$). Second, while the intervals $\frac{\Box f_{I,a,t}(J_\gamma)}{\Box f_{I,a,1}(J_\gamma)}$ overlap,[2] we refine the involved intervals J_γ. When refined, the precision of the interval is doubled, that is, if for some $k > 0$ the input width is 2^{-k} then the output width is less than 2^{-2k}.

---

[1] The assumption that the polynomials of the RUR of $\langle P, Q\rangle$ have integer coefficients satisfying the bitsize bound of Theorem 4.1.1 is without loss of generality since according to Proposition 4.3.1 one can compute such polynomials in a complexity that is less than that for computing the RUR.

[2] If 0 is in $\Box f_{I,a,1}(J_\gamma)$, the interval $\frac{\Box f_{I,a,t}(J_\gamma)}{\Box f_{I,a,1}(J_\gamma)}$ is equal to $\mathbb{R}$.

The complexity of computing isolating boxes is then the complexity of the initial isolation of the roots of $f_{I,a}$, plus the complexity of the loop. Since the width is quadratically smaller at each iteration, the number of iterations is in log of the log of the minimum width eventually reached by the intervals J_γ after refinement. We will show that for the boxes to be isolating, a worst-case refinement up to a width in 2^{-L} with $L = \widetilde{O}(d_r^2 + d_r\tau_r)$ for the roots of $f_{I,a}$ is sufficient. The overall cost of computing isolating intervals of the roots of $f_{I,a}$ and refining them up to a width in 2^{-L} with $L = \widetilde{O}(d_r^2+d_r\tau_r)$ is in $\widetilde{O}_B(d_r^3+d_r^2\tau_r)$ (Theorem 2.3.15). On the other hand, due to the dyadic representation of the computed isolating intervals, the bitsize of the endpoints these intervals is in $\widetilde{O}(d_r^2 + d_r\tau_r)$. In addition, since the refinement is adaptative with respect to each root, we will also show that the sum of the bitsizes of the endpoints of the computed isolating intervals J_γ, after refinement, is also in $\widetilde{O}(d_r^2 + d_r\tau_r)$.

Without counting the cost of refinements, the cost of the computation within one loop is then as follows: (a) The cost of evaluating the polynomials $f_{I,a,t}$ and $f_{I,a,1}$ of degree bounded by d_r and bitsize bounded by τ_r at all the (refined) intervals J_γ is in $\widetilde{O}_B(d_r^3+d_r^2\tau_r)$ by Theorem 2.3.4 and since the sum of the bitsizes of the endpoints of the (refined) intervals J_γ is in $\widetilde{O}(d_r^2+d_r\tau_r)$. (b) The cost of the overlap test for all the intervals $\frac{\Box f_{I,a,t}(J_\gamma)}{\Box f_{I,a,1}(J_\gamma)}$ is in $\widetilde{O}_B(d_r^3 + d_r^2\tau_r)$. Indeed, the bit complexity of the overlap test is the sum of the bitsizes of the endpoints of the intervals $\frac{\Box f_{I,a,t}(J_\gamma)}{\Box f_{I,a,1}(J_\gamma)}$ because when the intervals are sufficiently refined to be disjoint, their order is that of the corresponding intervals J_γ. Hence, we only need to perform comparisons between the endpoints of two intervals $\frac{\Box f_{I,a,t}(J_\gamma)}{\Box f_{I,a,1}(J_\gamma)}$ when the corresponding intervals J_γ are consecutive. Furthermore, the sum of the bitsizes of these interval endpoints is the sum over all J_γ of big $\widetilde{O}$ of τ_r plus d_r times the bitsize of the endpoints of J_γ (by Theorem 2.3.4), which sums up to $\widetilde{O}(d_r\tau_r+d_r(d_r^2+d_r\tau_r)) = \widetilde{O}(d_r^3+d_r^2\tau_r)$. The overall complexity of the algorithm is thus $\widetilde{O}_B(d_r^3 + d_r^2\tau_r)$ plus the complexity of the refinements which is also in $\widetilde{O}_B(d_r^3 + d_r^2\tau_r)$.

It remains to prove that (1) a refinement up to a width in 2^{-L} with $L = \widetilde{O}(d_r^2 + d_r\tau_r)$ for the isolating intervals of roots of $f_{I,a}$ is enough to compute isolating boxes, and (2) the sum of the bitsizes of the endpoints of the isolating intervals computed by this algorithm is in $\widetilde{O}(d_r^2 + d_r\tau_r)$.

A sufficient condition for the boxes to be isolating is that the intervals $\frac{\Box f_{I,a,t}(J_\gamma)}{\Box f_{I,a,1}(J_\gamma)}$ do not overlap. Since each of these intervals always contains the corresponding root γ of $f_{I,a}$, they do not overlap if their width is smaller than half the local separating bound $sep(\gamma)$. We then study the width of such an interval with respect to the width of the input interval J_γ.

One has $w(\frac{\Box f_{I,a,t}(J_\gamma)}{\Box f_{I,a,1}(J_\gamma)}) \leqslant 2\max_{y\in\frac{\Box f_{I,a,t}(J_\gamma)}{\Box f_{I,a,1}(J_\gamma)}} |\frac{f_{I,a,t}(\gamma)}{f_{I,a,1}(\gamma)} - y|$ and we can write $y = \frac{f_{I,a,t}(\gamma)+e_t}{f_{I,a,1}(\gamma)+e_1}$ with e_t, e_1 such that $f_{I,a,t}(\gamma) + e_t \in \Box f_{I,a,t}(J_\gamma)$ and $f_{I,a,1}(\gamma) +$

$e_1 \in \Box f_{I,a,1}(J_\gamma)$ thus

$$\begin{aligned}
&\left|\frac{f_{I,a,t}(\gamma)}{f_{I,a,1}(\gamma)} - \frac{f_{I,a,t}(\gamma)+e_t}{f_{I,a,1}(\gamma)+e_1}\right| \\
&= \frac{1}{|f_{I,a,1}(\gamma)+e_1|}\left|\frac{f_{I,a,t}(\gamma)(f_{I,a,1}(\gamma)+e_1)}{f_{I,a,1}(\gamma)} - f_{I,a,t}(\gamma) - e_t\right| \\
&= \frac{1}{|f_{I,a,1}(\gamma)+e_1|}\left|\frac{f_{I,a,t}(\gamma)}{f_{I,a,1}(\gamma)}e_1 - e_t\right| \\
&= \frac{1}{|f_{I,a,1}(\gamma)+e_1|}\left|\gamma e_1 - e_t\right| \\
&\leqslant \frac{1}{|f_{I,a,1}(\gamma)+e_1|}\max(|e_1|,|e_t|)\,(1+|\gamma|) \\
&\leqslant \frac{2}{|f_{I,a,1}(\gamma)+e_1|}\max(|e_1|,|e_t|)\max(1,|\gamma|).
\end{aligned}$$

We can now apply Theorem 2.3.16 to e_t and e_1 which gives $\max(|e_1|,|e_t|) \leqslant 2^{d_r+\tau_r}\max(1,|\gamma|)^{d_r-1}w(J_\gamma)$. In addition, for latter calculation to make sense, the denominator $f_{I,a,1}(\gamma)+e_1$ must not vanish. By the definition of the RUR, $f_{I,a,1}(\gamma)$ cannot vanish and a sufficient condition to ensure this property is that $w(J_\gamma) \leqslant \frac{|f_{I,a,1}(\gamma)|/2}{2^{d_r+\tau_r}\max(1,|\gamma|)^{d_r-1}}$; indeed this implies that $|e_1| \leqslant |f_{I,a,1}(\gamma)|/2$ and $|f_{I,a,1}(\gamma)+e_1| \geqslant |f_{I,a,1}(\gamma)| - |e_1| \geqslant |f_{I,a,1}(\gamma)|/2$. Putting everything together yields

$$w\left(\frac{\Box f_{I,a,t}(J_\gamma)}{\Box f_{I,a,1}(J_\gamma)}\right) \leqslant \frac{2^{d_r+\tau_r+3}\max(1,|\gamma|)^{d_r}}{|f_{I,a,1}(\gamma)|}w(J_\gamma).$$

Recall that to avoid overlaps it is sufficient that $w(\frac{\Box f_{I,a,t}(J_\gamma)}{\Box f_{I,a,1}(J_\gamma)}) < \frac{sep(\gamma)}{2}$, so it is sufficient that

$$\frac{2^{d_r+\tau_r+3}\max(1,|\gamma|)^{d_r}}{|f_{I,a,1}(\gamma)|}w(J_\gamma) < \frac{sep(\gamma)}{2}$$

or, equivalently,

$$w(J_\gamma) < \frac{|f_{I,a,1}(\gamma)|sep(\gamma)}{2^{d_r+\tau_r+4}\max(1,|\gamma|)^{d_r}}.$$

According to Theorem 2.3.12, $\max(1,|\gamma|) = 2^{O(\tau_r)}$, Theorem 2.3.13 yields $sep_\gamma = 2^{-\widetilde{O}(d_r\tau_r+d_r^2)}$ and Theorem 2.3.14 yields $|f_{I,a,1}(\gamma)| = 2^{-O(d_r\tau_r)}$. A sufficient condition for non-overlapping is thus that $w(J_\gamma) = 2^{-\widetilde{O}(d_r\tau_r+d_r^2)}$ or in other words that a refinement of the roots of $f_{I,a}$ up to a precision in $L = \widetilde{O}(d_r\tau_r+d_r^2)$ is sufficient.[3]

---

[3] Note that the requirement that the denominator $f_{I,a,1}(\gamma)+e_1$ must not vanish, which is ensured by requiring $w(J_\gamma) \leqslant \frac{|f_{I,a,1}(\gamma)|/2}{2^{d_r+\tau_r}\max(1,|\gamma|)^{d_r-1}}$, gives the same asymptotic behavior.

For bounding the cost of the evaluations of the RUR polynomials on the computed isolating intervals, an amortized bound of the sum of the bitsizes of the interval endpoints is needed. Due to the dyadic interval representation, the bitsize of this sum is of the same order as that of the product of the widths of all intervals. For all the roots together:

$$\prod w(J_\gamma) < \frac{\prod |f_{I,a,1}(\gamma)| \prod sep(\gamma)}{2^{d_r(d_r+\tau_r+4)} \prod \max(1, |\gamma|)^{d_r}}$$

and the application of the amortized bounds of Theorems 2.3.12, 2.3.13 and 2.3.14 yields that the sum of the bitsizes of the interval endpoints is in $\widetilde{O}(d_r\tau_r + d_r^2)$. □

Applying the previous theorem to the RURs computed in Chapter 4 or Chapter 5 we obtain the following result.

Theorem 6.1.2. *Given a RUR of $\langle P, Q\rangle$, isolating boxes for the solutions of $\langle P, Q\rangle$ can be computed in $\widetilde{O}_B(d^6 + d^5\tau)$ bit operations, where d bounds the total degree of P and Q, and τ bounds the bitsize of their coefficients. The vertices of these boxes have bitsize in $\widetilde{O}(d^4 + d^3\tau)$.*

Proof. Depending on the algorithm we use for computing the RUR of $\langle P, Q\rangle$, the RUR is given either as a single RUR (Chapter 4) or as a set of RURs (Chapter 5). In the first case, the result follows straightforwardly using the bounds of Theorem 4.1.1. For the second case, it is sufficient to notice that each RUR resulting from the decomposition satisfies the bitsize bound of Proposition 4.3.1 since the corresponding ideal contains the ideal $\langle P, Q\rangle$ by Lemma 5.3.2, and that the sum of their degrees is bounded by the Bézout's bound of $\langle P, Q\rangle$ that is, d^2 also by Lemma 5.3.2. Hence, denoting by d_i and τ_i the degrees and bitsizes of the RURs resulting from the decomposition, we have $\sum_{i=1}^{d} d_i \leqslant d^2$ and $\tau_i \in \widetilde{O}(d^2 + d\tau)$ thus $\sum_{i=1}^{d} d_i^3 + d_i^2\tau_i$ is in $\widetilde{O}(d^6 + d^5\tau)$ and $\sum_{i=1}^{d} d_i^2 + d_i\tau_i$ is in $\widetilde{O}(d^4 + d^3\tau)$, which concludes the proof. □

6.2 Sign of a polynomial at the solutions of a system

The results presented in this section are part of an article published in the ISSAC 2013 conference [Bouzidi 2013b].

This section addresses the problem of computing the sign (+, − or 0) of a given polynomial $F \in \mathbb{Z}[X, Y]$ at the solutions of a bivariate system defined by an ideal I. Let $\{f_{I,a}, f_{I,a,1}, f_{I,a,X}, f_{I,a,Y}\}$ be the RUR of I. As in the previous section, the degree and the bitsize of this RUR are denoted respectively by d_r

and τ_r and we denote by d and τ the degree and the bitsize of F.[4] We also suppose for simplicity that $d = O(d_r)$ and $\tau = O(\tau_r)$. We first describe a naive RUR-based *sign_ at* algorithm for computing the sign at one real solution of the system, which runs in $\widetilde{O}_B(d^9 + d^8\tau)$ time. Then, using properties of generalized Sturm sequences, we analyze a more efficient algorithm that runs in $\widetilde{O}_B(d^8 + d^7\tau)$ time. We also show that the sign of F at the $O(d^2)$ solutions of the system can be computed in only $O(d)$ times that for one real solution.

Once the RUR $\{f_{I,a}, f_{I,a,1}, f_{I,a,X}, f_{I,a,Y}\}$ of $I = \langle P, Q\rangle$ is computed, we can use it to translate a bivariate sign computation into a univariate sign computation. Indeed, let $F(X,Y)$ be the polynomial to be evaluated at the solution (α,β) of I that is the image of the root γ of $f_{I,a}$ by the RUR mapping. We first define the polynomial $f_F(T)$ roughly as the numerator of the rational fraction obtained by substituting $X = \frac{f_{I,a,X}(T)}{f_{I,a,1}(T)}$ and $Y = \frac{f_{I,a,Y}(T)}{f_{I,a,1}(T)}$ in the polynomial $F(X,Y)$, so that the sign of $F(\alpha,\beta)$ is the same as that of $f_F(\gamma)$.

Lemma 6.2.1. *The polynomial* $f_F(T) = f_{I,a,1}^d(T)F(T-aY,Y)$, *with* $Y = \frac{f_{I,a,Y}(T)}{f_{I,a,1}(T)}$, *has degree* $O(dd_r)$, *bitsize in* $\widetilde{O}(d\tau_r)$, *and it can be computed with* $\widetilde{O}_B(d^3 d_r\tau_r)$ *bit operations. The sign of* F *at a real solution of* I *is equal to the sign of* f_F *at the corresponding root of* $f_{I,a}$ *via the mapping of the RUR.*

Proof. We first compute the polynomial $F(T-aY,Y)$ in the form $\sum_{i=0}^{d} a_i(T)Y^i$. Then, $f_F(T)$ is equal to $\sum_{i=0}^{d} a_i(T)f_{I,a,Y}(T)^i f_{I,a,1}(T)^{d-i}$. Consequently, computing an expanded form of $f_F(T)$ can be done by computing the $a_i(T)$, the powers $f_{I,a,Y}(T)^i$ and $f_{I,a,1}(T)^i$, and their appropriate products and sum.

Computing $a_i(T)$. According to Lemma 3.2.1, $P(T-SY,Y)$ can be expanded with $\widetilde{O}_B(d^4 + d^3\tau)$ bit operations and its bitsize is in $\widetilde{O}(d+\tau)$. These bounds also apply to $F(T-SY,Y)$ and we deduce $F(T-aY,Y)$ by substituting S by a. Writing $F(T-SY,Y) = \sum_{i=0}^{d} f_i(T,Y)S^i$, the computation of $F(T-aY,Y)$ can be done by computing and summing the $f_i(T,Y)a^i$. Since a has bitsize in $O(\log d)$ by hypothesis, a^i has bitsize in $O(d\log d) \subseteq \widetilde{O}(d)$, and computing all the a^i can be done with $\widetilde{O}_B(d^2)$ bit operations. For each a^i, computing $f_i(T,Y)a^i$ can be done with $O(d^2)$ multiplications between integers of bitsize in $\widetilde{O}(d+\tau)$, and thus with $\widetilde{O}_B(d^2(d+\tau))$ bit operations. Thus, computing all the $f_i(T,Y)a^i$ can be done with $\widetilde{O}_B(d^3(d+\tau))$ bit operations, and summing, for every of the $O(d^2)$ monomials in (T,Y), d coefficients (corresponding to every i) of bitsize in $\widetilde{O}(d+\tau)$ can also be done with $\widetilde{O}_B(d^3(d+\tau))$ bit operations, in total. It follows that, $F(T-aY,Y)$ and thus all the $a_i(T)$ can be computed with $\widetilde{O}_B(d^4 + d^3\tau)$ bit operations.

---

[4] We assume without loss of generality that the degree d is *even*.

Computing $f_{I,a,Y}(T)^i$ and $f_{I,a,1}(T)^i$. $f_{I,a,Y}(T)^i$ has degree in $O(dd_r)$ and bitsize in $\widetilde{O}(d\tau_r)$. Computing all the $f_{I,a,Y}(T)^i$ can be done with $O(d)$ multiplications between these polynomials. Every multiplication can be done with a bit complexity that is softly linear in the product of the maximum degrees and maximum bitsizes (Theorem 2.3.1), thus all the multiplications can be done with $\widetilde{O}_B(d^3d_r\tau_r)$ bit operations in total. It follows that all the $f_{I,a,Y}(T)^i$, and similarly all the $f_{I,a,1}(T)^i$, can be computed using $\widetilde{O}_B(d^3d_r\tau_r)$ bit operations and their bitsize is in $\widetilde{O}(d\tau_r)$.

Computing $f_F(T)$. Computing $a_i(T)f_{I,a,Y}(T)^i f_{I,a,1}(T)^{d-i}$, for $i = 0, \dots, d$, amounts to multiplying $O(d)$ times, univariate polynomials of degree $O(dd_r)$ and bitsize $\widetilde{O}(d\tau_r)$, which can be done, similarly as above, with $\widetilde{O}(d^3d_r\tau_r)$ bit operations. Finally, their sum is the sum of d univariate polynomials of degree $O(dd_r)$ and bitsize $\widetilde{O}(d\tau_r)$, which can also be computed within the same bit complexity. Hence, $f_F(T)$ can be computed with $\widetilde{O}_B(d^3d_r\tau_r)$ bit operations and its coefficients have bitsize in $\widetilde{O}(d\tau_r)$.

Signs of F and f_F. It remains to show that the sign of F at a real solution of I is the sign of f_F at the corresponding root of $f_{I,a}$ via the mapping of the RUR. By Definition 2.2.1, there is a one-to-one mapping between the roots of $f_{I,a}$ and those of I that maps a root γ of $f_{I,a}$ to a solution $(\alpha, \beta) = (\frac{f_{I,a,X}(\gamma)}{f_{I,a,1}(\gamma)}, \frac{f_{I,a,Y}(\gamma)}{f_{I,a,1}(\gamma)})$ of I such that $\gamma = \alpha + a\beta$ and $f_{I,a,1}(\gamma) \neq 0$. For any such pair of γ and (α, β), $f_F(\gamma) = f_{I,a,1}^d(\gamma)F(\gamma - a\frac{f_{I,a,Y}(\gamma)}{f_{I,a,1}(\gamma)}, \frac{f_{I,a,Y}(\gamma)}{f_{I,a,1}(\gamma)})$ by definition of $f_F(T)$, and thus $f_F(\gamma) = f_{I,a,1}^d(\gamma)F(\alpha, \beta)$. It follows that $f_F(\gamma)$ and $F(\alpha, \beta)$ have the same sign since $f_{I,a,1}(\gamma) \neq 0$ and d is even by hypothesis. □

Naive algorithm. The knowledge of a RUR $\{f_{I,a}, f_{I,a,1}, f_{I,a,X}, f_{I,a,Y}\}$ of I yields a straightforward algorithm for computing the sign of F at a real solution of I. Indeed, it is sufficient to isolate the real roots of $f_{I,a}$, so that the intervals are also isolating for $f_{I,a}f_F$, and then to evaluate the sign of $\overline{f_F}$ at the endpoints of these isolating intervals. We analyze the complexity of this straightforward algorithm before describing our more subtle and more efficient algorithm. We provide this analysis for several reasons: first it answers a natural question, second it shows that even a RUR-based naive algorithm performs better than the state of the art.

Lemma 6.2.2. *Given a RUR $\{f_{I,a}, f_{I,a,1}, f_{I,a,X}, f_{I,a,Y}\}$ of I and an isolating interval for a real root γ of $f_{I,a}$, the sign of F at the real solution of I that corresponds to γ can be computed with $\widetilde{O}_B(d^3d_r^2\tau_r + d_r^3)$ bit operations.*

Proof. By Lemma 6.2.1, f_F has degree $O(dd_r)$ and bitsize $\widetilde{O}(d\tau_r)$, and it can be computed with $\widetilde{O}_B(d^2d_r\tau_r)$ bit operations. Thus the product $f_F\, f_{I,a}$ has degree $O(dd_r)$ and bitsize $\widetilde{O}(d\tau_r)$. By Theorem 2.3.13, the root separation bound of $f_F\, f_{I,a}$ has bitsize $\widetilde{O}(d^2 d_r \tau_r)$. We refine the isolating interval of γ for $f_{I,a}$ to a width less than the root separation bound of $f_F\, f_{I,a}$, which can be done with $\widetilde{O}_B(d_r^3 + d_r^2\tau_r + d_r(d^2 d_r \tau_r)) = \widetilde{O}_B(d^2 d_r^2 \tau_r + d_r^3)$ bit operations according to Theorem 2.3.15. Furthermore, we can ensure that the new interval has rational endpoints with bitsize $\widetilde{O}(d^2 d_r \tau_r)$, similarly as in the proof of Theorem 6.1.1. On the other hand, by Theorem 2.3.10, since f_F has bitsize $\widetilde{O}(d\tau_r)$, its squarefree part $\overline{f_F}$ can be computed in complexity $\widetilde{O}_B((dd_r)^2(d\tau_r)) = \widetilde{O}_B(d^2 d_r^2 \tau_r)$ and it has bitsize in $\widetilde{O}_B(d\tau_r + dd_r)$. It then follows from Theorem 2.3.4 that the evaluation of $\overline{f_F}$ at the boundaries of the refined interval can be done with $\widetilde{O}_B(d^3 d_r^2 \tau_r)$ bit operations which concludes the proof by Lemma 6.2.1. □

Improved algorithm. Our more subtle algorithm is, in essence the one presented by Diochnos et al. for evaluating the sign of a univariate polynomial (here f_F) at the roots of a squarefree univariate polynomial (here $\overline{f_{I,a}}$) [Diochnos 2009, Corollary 5]. The idea of this algorithm comes originally from [Lickteig 2001], where the Cauchy index of two polynomials is computed by means of sign variations of a particular remainder sequence called the Sylvester-Habicht sequence. In [Diochnos 2009], this approach is slightly adapted to deduce the sign from the Cauchy index ([Yap 2000, Theorem 7.3]) and the bit complexity is given in terms of the two initial degrees and bitsizes. Unfortunately, the corresponding proof is problematic because the authors refer to two complexity results for computing parts of the Sylvester-Habicht sequences and none of them actually applies.[5] Following the spirit of their approach, we present in Lemma 6.2.3 a new (weaker) complexity result for evaluating the sign of a univariate polynomial at the roots of a squarefree univariate polynomial. This result is used to derive the bit complexity of evaluating the sign of a bivariate polynomial at the roots of the system. For clarity, we postpone the proof of this lemma to Section 6.2.1 after Theorem 6.2.4.

Lemma 6.2.3. *Let $f \in \mathbb{Z}[X]$ be a squarefree polynomial of degree d_f and bitsize τ_f, and (a, b) be an isolating interval of one of its real roots γ with a and b distinct rationals of bitsize in $\widetilde{O}(d_f \tau_f)$ and $f(a)f(b) \neq 0$. Let $g \in \mathbb{Z}[X]$ be of degree d_g and bitsize τ_g. The sign of $g(\gamma)$ can be computed in*

---

[5] Precisely, their proof is based on their Proposition 1 which claims, based on [Lickteig 2001] and [Reischert 1997] that given two polynomials f and g of degree $p > q$ and bitsize in $O(\tau)$, any of their polynomial subresultants as well as the whole quotient chain corresponding to the subresultant sequence can be computed with $\widetilde{O}_B(pq\tau)$ bit operations. However, in [Lickteig 2001] the complexity results are not stated in terms of p and q but only in terms of the maximum degree while in [Reischert 1997], the result assumes that the $(q-1)^{th}$ subresultant of f and g is known.

$\widetilde{O}_B((d_f^3+d_g^2)\tau_f+(d_f^2+d_fd_g)\tau_g)$ bit operations. The sign of g at all the real roots of f can be computed with $\widetilde{O}_B((d_f^3+d_f^2d_g+d_g^2)\tau_f+(d_f^3+d_fd_g)\tau_g)$ bit operations.

Theorem 6.2.4. *Given a RUR $\{f_{I,a}, f_{I,a,1}, f_{I,a,X}, f_{I,a,Y}\}$ of I, the sign of F at a real solution of I can be computed with $\widetilde{O}_B(d_r^3\tau_r+d^2d_r^2\tau_r)$ bit operations. The sign of F at all the solutions of I can be computed with $\widetilde{O}_B(dd_r^3\tau_r+d^2d_r^2\tau_r)$ bit operations.*

Proof. By Lemma 6.2.1, the sign of F at the real solutions of I, is equal to the sign of f_F at the corresponding roots of $f_{I,a}$, or equivalently at those of $\overline{f_{I,a}}$. Furthermore, f_F has degree $O(dd_r)$, bitsize in $\widetilde{O}(d\tau_r)$, and it can be computed with $\widetilde{O}_B(d^3d_r\tau_r)$ bit operations. On the other hand, by Theorem 2.3.10, the squarefree part of $f_{I,a}$ can be computed in $\widetilde{O}_B(d_r^2\tau_r)$ bit operations and it has bitsize in $\widetilde{O}(\tau_r+d_r)$. By Theorem 2.3.15 and 2.3.13, the isolating intervals (if not given) of $\overline{f_{I,a}}$ can be computed in $\widetilde{O}_B((d_r)^3+(d_r)^2(\tau_r+d_r))$ bit operations with intervals boundaries of bitsize satisfying the hypotheses of Lemma 6.2.3. Indeed, we can ensure during the isolation of the roots of $f=\overline{f_{I,a}}$ that the isolating intervals have endpoints with bitsize in $\widetilde{O}(d_r\tau_r)$, similarly as in the proof of Theorem 6.1.1. Applying Lemma 6.2.3 then concludes the proof. ☐

Applying the previous theorem to the RURs computed in Chapter 4 or Chapter 5 and using the same arguments as for Theorem 6.1.2, we obtain the following result.

Proposition 6.2.5. *Given a RUR of $\langle P,Q\rangle$ and assuming that the degree of P and Q is bounded by d and their bitsize is bounded by τ, the sign of F at a real solution of $\langle P,Q\rangle$ can be computed with $\widetilde{O}_B(d^8+d^7\tau)$ bit operations. The sign of F at all the solutions of I can be computed with $\widetilde{O}_B(d^9+d^8\tau)$ bit operations.*

Remark 6.2.6. *Proposition 6.2.5 also holds if the solutions of $I=\langle P,Q\rangle$ are described by the rational parameterization of Gonzalez-Vega and El Kahoui [Gonzalez-Vega 1996] instead of a RUR. Indeed, such parameterization is defined, in the worst case, by $\Theta(d)$ univariate polynomials f_i of degree d_{f_i} whose sum d_f is at most d^2, and by associated rational one-to-one mappings which are defined, as for the RUR, by polynomials of degree $O(d^2)$ and bitsize $O(d^2+d\tau)$. The result of Proposition 6.2.5 on the sign of F at* one *real solution of I thus trivially still holds. For the sign of F at* all *real solutions of I the result also holds from the following observation. In the proofs of Lemmas 6.2.7 and 6.2.3, the computation of one sequence of unevaluated Sylvester-Habicht transition matrices has complexity $\widetilde{O}_B(pH)$ (in proof of Lemma 6.2.7) where p is in $O(d_{f_i}+d_g)$ in the proof of Lemma 6.2.3. The* *sum of the pH over all i is thus* $O((d_f + dd_g)H)$ *instead of* $O((d_f + d_g)H)$ *as for the RUR. However,* d_gH *writes in the proof of Lemma 6.2.3 as* $\widetilde{O}(d_g((d_f + d_g)\tau_f + d_f(\tau_f + \tau_g))) = \widetilde{O}(d_fd_g(\tau_f + \tau_g) + d_g^2\tau_f)$ *which writes in the proof of Theorem 6.2.4 as* $\widetilde{O}(d^2d^3(d^3 + d^2\tau) + (d^3)^2(d^2 + d\tau)) = \widetilde{O}(d^8 + d^7\tau)$*. Thus multiplying this by d remains within the targeted bit complexity. On the other hand, the complexity of the evaluation phase in the proofs of Lemmas 6.2.7 and 6.2.3 does not increase when considering the representation of Gonzalez-Vega and El Kahoui instead of the RUR because the total complexity of the evaluations depends only on the number of solutions at which we evaluate the sign of the other polynomial and on the degree and bitsize of the polynomials involved (values which do not increase in Gonzalez-Vega and El Kahoui representation; only the number of polynomials is larger).*

6.2.1 Proof of Lemma 6.2.3

As shown in [Basu 2006, Theorem 2.61], the sign of $g(\gamma)$ is $V(SRemS(f, f'g; a, b))$ where $V(SRemS(P, Q; a, b))$ is the number of sign variations in the signed remainder sequence of P and Q evaluated at a minus the number of sign variations in this sequence evaluated at b (see Definition 1.7 in [Basu 2006] for the sequence and Notation 2.32 for the sign variation). On the other hand, for any P and Q such that $\deg(P) > \deg(Q)$ and $P(a)\,P(b) \neq 0$ or $Q(a)\,Q(b) \neq 0$, we have according to [Roy 1996, Theorems 3.2, 3.18 & Remarks 3.9, 3.25][6] that $V(SRemS(P, Q; a, b)) = W(SylH(P, Q; a, b))$ where $SylH$ is the Sylvester-Habicht sequence of P and Q, and W is the related sign variation function.[7] The following intermediate result is a consequence of an adaptation of [Lickteig 2001, Theorem 5.2] in the case where the polynomials P and Q have different degrees and bitsizes.

Lemma 6.2.7. *Let* P *and* Q *in* $\mathbb{Z}[X]$ *with* $deg(P) = p > q = deg(Q)$ *and bitsize respectively* τ_P, τ_Q*. If* a *and* b *are two rational numbers of bitsize bounded by* σ*, the computation of* $W(SylH(P, Q; a, b))$ *can be performed with* $\widetilde{O}_B((p + q^2)\sigma + p(p\tau_Q + q\tau_P))$ *bit operations.*

---

[6] The same result can be found directly stated, in French, in [Lombardi 1990, Theorem 4].

[7] The Sylvester-Habicht sequence, defined in [Basu 2006, §8.3.2.2] as the Signed Subresultant sequence, can be derived from the classical subresultant sequence [El Kahoui 2003] by multiplying the two starting subresultants by +1 the next two by −1 and so on. W is defined as the usual sign variation with the following modification for groups of two consecutive zeros: count *one* sign variation for the groups $[+, 0, 0, -]$ and $[-, 0, 0, +]$, and *two* sign variations for the $[+, 0, 0, +]$ and $[-, 0, 0, -]$ (see [Basu 2006, §9.1.3 Notation 9.11]).

Moreover, if a_ℓ and b_ℓ, $1 \leqslant \ell \leqslant u$, are rational numbers of bitsizes that sum to σ, the computation of $W(SylH(P, Q; a_\ell, b_\ell))$ can be performed for all ℓ with $\widetilde{O}_B((p + q^2)\sigma + (p + qu)(p\tau_Q + q\tau_P) + pu\tau_P)$ bit operations.

Proof. Following the algorithm in [Lickteig 2001], we first compute the consecutive Sylvester-Habicht transition matrices of P and Q denoted by $\mathcal{N}_{j,i}$ with $0 \leqslant j < i \leqslant p$. These matrices link consecutive regular couples[8] (Sh_i, Sh_{i-1}) and (Sh_j, Sh_{j-1}) in the Sylvester-Habicht sequence as follows:

$$\begin{pmatrix} Sh_j \\ Sh_{j-1} \end{pmatrix} = \mathcal{N}_{j,i} \begin{pmatrix} Sh_i \\ Sh_{i-1} \end{pmatrix} \text{ such that } i \leqslant p \text{ and } (Sh_p, Sh_{p-1}) = (P, Q). \quad (6.1)$$

According to [Lickteig 2001, Theorem. 5.2 & Corollary 5.2], computing all the matrices $\mathcal{N}_{j,i}$ of P and Q can be done with $\widetilde{O}_B(pH)$ bit operations, where $H \in \widetilde{O}(q\tau_P + p\tau_Q)$ is an upper bound on the bitsize appearing in the computations given by Hadamard's inequality.

We evaluate the Sylvester-Habicht sequence at a rational a by first evaluating P, Q, and all the matrices $\mathcal{N}_{j,i}$ at a, and then by applying iteratively the above formula. Doing the same at b yields $W(SylH(P, Q; a, b))$.

First, note that the evaluation of $P(a)$ and $Q(a)$ can be done with $\widetilde{O}_B(p(\tau_P + \sigma))$ plus $\widetilde{O}_B(q(\tau_Q + \sigma))$, that is $\widetilde{O}_B(p(\tau_P + \tau_Q + \sigma))$ bit operations (since $p > q$), by Theorem 2.3.4. The polynomials appearing in the matrices $\mathcal{N}_{j,i}$ have bitsize at most H and the sum of their degrees is equal to p [Lickteig 2001, Corollary 4.3].[9] Thus, all $\mathcal{N}_{j,i}(a)$ have bitsize $\widetilde{O}(p\sigma + H)$ and they can be computed in a total of $\widetilde{O}_B(p(\sigma + H))$ bit operations, by Theorem 2.3.4. Moreover, by considering the matrices $\mathcal{N}_{j,i}$ other than the first one $\mathcal{N}_{k,p}$, as the consecutive transition matrices of the Sylvester-Habicht sequence of the first regular couple (Sh_k, Sh_{k-1}) after (Sh_p, Sh_{p-1}), we have that the polynomials appearing in these matrices have the sum of their degrees equal to that of Sh_k which is at most q (since $k \leqslant p-1$ and $Sh_{p-1} = Q$). Thus, except the first one $\mathcal{N}_{k,p}(a)$, all evaluated matrices $\mathcal{N}_{j,i}(a)$ have bitsize $\widetilde{O}(q\sigma + H)$ and they can be computed in a total of $\widetilde{O}_B(q(\sigma + H))$ bit operations.

We now apply iteratively Equation (6.1) for computing all the $Sh_i(a)$. Since all Sylvester-Habicht polynomials have bitsize at most H and degree at most q except the first one $Sh_p = P$, the bitsize of $Sh_{i<p}(a)$ is in $O(q\sigma + H)$ and that of $Sh_p(a)$ is in $O(p\sigma + \tau_P)$. Given $P(a)$, $Q(a)$ and all $\mathcal{N}_{j,i}(a)$, it follows

---

[8] Regular couples in the Sylvester-Habicht sequence are the nonzero Sylvester-Habicht polynomials (Sh_i, Sh_{i-1}) such that $\deg(Sh_i) > \deg(Sh_{i-1})$.

[9] [Lickteig 2001, Corollary 4.3] states that consecutive Sylvester-Habicht transition matrices consist of one zero, two integers and a polynomial which is, up to a coefficient, the quotient of the division of two consecutive Sylvester-Habicht polynomials. These polynomials being proportional to polynomials in the remainder sequence of (P, Q), the sum of the degrees of their quotients is equal to the degree of P.

from their bitsizes that we can compute iteratively the $Sh_i(a)$ in time $\widetilde{O}_B(p\sigma + H)$ for the first regular couple after $(Sh_p, Sh_{p-1}) = (P, Q)$ and in time $\widetilde{O}_B(q\sigma + H)$ for each of the others. Thus, for computing of $W(SylH(P, Q; a, b))$, the initial computation of all $\mathcal{N}_{j,i}$ takes $\widetilde{O}_B(pH)$ bit operations and the evaluation phase takes $\widetilde{O}_B(p(\tau_P + \tau_Q + \sigma))$ plus $\widetilde{O}_B(p(\sigma + H) + q(q\sigma + H))$ bit operations, which gives a total of $\widetilde{O}_B(p(\sigma + H) + q^2\sigma)$ bit operations.

We now consider the case of computing $W(SylH(P, Q; a_\ell, b_\ell))$ for $1 \leqslant \ell \leqslant u$. We slightly change the above algorithm as follows. We only change the way to evaluate the first regular couple (Sh_k, Sh_{k-1}) after (Sh_p, Sh_{p-1}) at the a_ℓ (and b_ℓ). Once the matrices $\mathcal{N}_{j,i}$ have been computed, we compute the (non-evaluated) first regular couple $(Sh_k, Sh_{k-1}) = \mathcal{N}_{k,p}(Sh_p, Sh_{p-1})$. Since the polynomials in $\mathcal{N}_{k,p}$ have degree at most p and bitsize at most H, the couple (Sh_k, Sh_{k-1}) can be computed in $\widetilde{O}_B(p(H + \tau_P + \tau_Q)) = \widetilde{O}_B(pH)$ time [von zur Gathen 2003, Corollary 8.27]. As noted above, Sh_k, and thus also Sh_{k-1}, have degree at most q and they have bitsize at most H, so they can be evaluated at a given a_ℓ in time $\widetilde{O}_B(q(\sigma_\ell + H))$ where σ_ℓ is the bitsize of a_ℓ. Now, the polynomials appearing in the matrices $\mathcal{N}_{j,i}$, other than the first one $\mathcal{N}_{k,p}$, have bitsize at most H and the sum of their degrees is at most q, so similarly as above, all the $\mathcal{N}_{j,i}(a_\ell)$, except $\mathcal{N}_{k,p}(a_\ell)$, can be computed in total bit complexity $\widetilde{O}_B(q(\sigma_\ell + H))$. Then, we compute as above each of the other regular couples evaluated at a_ℓ in time $\widetilde{O}_B(q\sigma_\ell + H)$. Hence, the initial computation of all $\mathcal{N}_{j,i}$ and of (Sh_k, Sh_{k-1}) takes $\widetilde{O}_B(pH)$ bit operations and the evaluation phase at all the a_ℓ takes the sum over ℓ, $1 \leqslant \ell \leqslant u$, of $\widetilde{O}_B(p(\tau_P + \tau_Q + \sigma_\ell))$ plus $\widetilde{O}_B(q(\sigma_\ell + H) + q(q\sigma_\ell + H))$ bit operations, that is $\widetilde{O}_B(p(\tau_P + \tau_Q) + (p + q^2)\sigma_\ell + qH)$ which sums to $\widetilde{O}_B(pu(\tau_P + \tau_Q) + (p + q^2)\sigma + quH)$. Hence the total bit complexity for computing all the $W(SylH(P, Q; a_\ell, b_\ell))$ for $1 \leqslant \ell \leqslant u$ is $\widetilde{O}_B((p + q^2)\sigma + (p + qu)H + pu\tau_P)$ which concludes the proof. □

Proof of Lemma 6.2.3. We may assume that g has degree greater than one since, if g is a constant the problem is trivial and, if $g(X) = cX - d$, then the sign of $g(\gamma)$ follows from (i) the sign of c if $\frac{d}{c} \notin (a, b)$ and from (ii) the signs of c, $f(a)$, and $f(\frac{d}{c})$ if $\frac{d}{c} \in (a, b)$; indeed, the signs of $f(a) \neq 0$ and $f(\frac{d}{c})$ determine whether γ lies in $(a, \frac{d}{c})$, $\{\frac{d}{c}\}$, or $(\frac{d}{c}, b)$. Hence, when g has degree one, the sign of $g(\gamma)$ can be computed with $\widetilde{O}_B(d_f(\tau_g + d_f\tau_f))$ bit operations according to Theorem 2.3.4.

Recall that the sign of $g(\gamma)$ is $V(SRemS(f, f'g; a, b))$ [Basu 2006, Theorem 2.61]. When g has degree greater than one, we cannot directly apply Lemma 6.2.7 since $\deg(f) < \deg(f'g)$. However, knowing the sign of f and $f'g$ at a and b and noticing that their signed remainder sequence starts with $[f, f'g, -f, -rem(f'g, -f), \ldots]$, we can easily compute the value c such that $V(SRemS(f, f'g; a, b)) = V(SRemS(f'g, -f; a, b)) + c$. Furthermore, as observed at the beginning of this section and since $f(a)f(b) \neq 0$ by hypothesis, $V(SRemS(f'g, -f; a, b)) = W(SylH(f'g, -f;\ a, b))$. We can now apply Lemma 6.2.7 which thus yields the sign of $g(\gamma)$ with a bit complexity in $\widetilde{O}_B((p+q^2)\sigma+p(p\tau_Q+q\tau_P))$ which simplifies into $\widetilde{O}_B((d_f^3+d_g^2)\tau_f+(d_f^2+d_f d_g)\tau_g)$.

For the sign of g at all the real roots of f, isolating intervals of these roots can be computed in complexity $\widetilde{O}_B(d_f^3 + d_f^2\tau_f)$ (see Theorem 2.3.15) such that the bitsizes of the interval boundaries sum up to $\widetilde{O}(d_f^2 + d_f\tau_f)$ (a consequence of Davenport-Mahler-Mignotte bound, see e.g. [Diochnos 2009, Lemma 6]). Similarly as for one root, Lemma 6.2.7 then yields that the sign of g at all the real roots of f can be computed with a bit complexity in $\widetilde{O}_B((p + q^2)\sigma + (p + qu)(p\tau_Q + q\tau_P) + pu\tau_P)$ which writes as $\widetilde{O}_B((d_f + d_g + d_f^2)d_f\tau_f + (d_f + d_g + d_f^2)((d_f + d_g)\tau_f + d_f(\tau_f + \tau_g)) + (d_f + d_g)d_f(\tau_g + \tau_f))$ and simplifies into $\widetilde{O}_B((d_f^3 + d_f^2 d_g + d_g^2)\tau_f + (d_f^3 + d_f d_g)\tau_g)$ bit operations. □

6.3 Over-constrained systems

So far, we focused on systems defined by exactly two coprime polynomials. We now extend our results to compute rational parameterizations of zero-dimensional systems defined with additional equality or inequality constraints. We assume given $RUR_{I,a} = \{f_{I,a}, f_{I,a,1}, f_{I,a,X}, f_{I,a,Y}\}$ the RUR of an ideal I associated to the separating form $X+aY$, we also assume that the polynomials of this RUR have degree bounded by d_r and bitsize bounded by τ_r. Given another polynomial $F \in \mathbb{Z}[X, Y]$, we have seen in the previous section how to compute the sign of F at the solutions of I. With a similar approach, we now explain how to split $RUR_{I,a}$ according to whether F vanishes or not at the solutions of I.

Let $F \in \mathbb{Z}[X, Y]$ be of total degree at most d and maximum bitsize τ. Identifying the roots of $f_{I,a}$ with the solutions of the system I via the RUR, let $f_{F=0}$ (resp. $f_{F\neq 0}$) be the squarefree factor of $f_{I,a}$ such that its roots are exactly the solutions of the system I at which the polynomial F vanishes (resp. does not vanish).

Theorem 6.3.1. *Given $RUR_{I,a}$, the bit complexity of computing $f_{F=0}$ (resp. $f_{F\neq 0}$) is in $\widetilde{O}_B(d^2d_r^2\tau_r+d^3d_r\tau_r)$ and these polynomials have bitsize in $\widetilde{O}(d_r+\tau_r)$.*

Proof. The polynomial f_F (not to be confused with $f_{F=0}$ or $f_{F\neq 0}$), as defined in Lemma 6.2.1, has the same sign as F at the real solutions of the system I. The same holds for complex solutions by considering the "sign" as zero or nonzero. The roots of the squarefree polynomial $f_{F=0} = \gcd(\overline{f_{I,a}}, f_F)$ thus are the $\alpha + a\beta$ with (α, β) solution of I and $F(\alpha, \beta) = 0$. The polynomial $f_{F\neq 0}$

defined as the gcd-free part of $\overline{f_{I,a}}$ with respect to f_F is also squarefree and encodes the solutions such that $F(\alpha, \beta) \neq 0$.

According to Lemma 6.2.1 and the proof of Theorem 6.2.4, f_F and $\overline{f_{I,a}}$ can be computed in, respectively, $\widetilde{O}_B(d^3 d_r \tau_r)$ and $\widetilde{O}_B(d_r^2 \tau_r)$ bit operations. Moreover, these integer polynomials have, respectively, bitsize $\widetilde{O}(d\tau_r)$ and $\widetilde{O}(d_r + \tau_r)$ and degree $O(dd_r)$ and $O(d_r)$. Thus, by Theorem 2.3.11, their gcd and the gcd-free part of $\overline{f_{I,a}}$ with respect to f_F, i.e. $f_{F=0}$ and $f_{F\neq 0}$, can be computed with $\widetilde{O}_B(d^2 d_r^2 \tau_r)$ bit operations and they have bitsize in $\widetilde{O}(d_r + \tau_r)$. □

For several equality or inequality constraints, iterating this splitting process gives a parameterization of the corresponding set of constraints. It is worth noticing that the set of polynomials $\{f_{F=0}, f_{I,a,1}, f_{I,a,X}, f_{I,a,Y}\}$ defines a rational parameterization of the solutions of the ideal $I + \langle F \rangle$, but this is not a RUR of this ideal (in the sense of Definition 2.2.1). First, because multiplicities are lost in the splitting process and second because the coordinate polynomials of the parameterization are still those of the ideal I. Still, it is possible to compute a RUR of the radical of the corresponding ideal (and similarly for the ideal corresponding to $F \neq 0$):

Theorem 6.3.2. *Given $RUR_{I,a}$ and $F \in \mathbb{Z}[X,Y]$ of total degree at most d and maximum bitsize τ, the bit complexity of computing the RUR of the radical of the ideal $I + \langle F \rangle$ is in $\widetilde{O}_B(d_r^4 + d_r^3 \tau_r)$.*

Proof. Denote by J the radical of the ideal $I + \langle F \rangle$. The polynomial $f_{F=0}$ computed in Theorem 6.3.1 is the first polynomial $f_{J,a}$ of $RUR_{J,a}$. Indeed, it vanishes at the solutions of this ideal (with identification of the roots of $f_{J,a}$ with the solutions of the system J) and it is squarefree. Then Proposition 4.2.1 yields that $f_{J,a,1}$ is the gcd-free part of $f'_{J,a}$ with respect to $f_{J,a}$. The latter can be computed in $\widetilde{O}_B(d_r^3 + d_r^2 \tau_r)$ according to Theorem 2.3.10.

According to Definition 2.2.1 of a RUR, the X-coordinates of the solutions of J are given by the polynomial fraction $\frac{f_{J,a,X}}{f_{J,a,1}}$ at the roots of $f_{J,a}$. On the other hand, the solutions of J, seen as solutions of I, have their X-coordinates defined by the polynomial fraction $\frac{f_{I,a,X}}{f_{I,a,1}}$. This thus implies that$f_{J,a,X} = f_{I,a,1}^{-1} f_{I,a,X} f_{J,a,1}$ modulo $f_{J,a}$. The computation of $f_{I,a,1}^{-1}$ together with the multiplication with other polynomials of the RUR can be done in $\widetilde{O}_B(d_r^3 + d_r^2 \tau_r)$ time ([von zur Gathen 2003, Corollary 11.11]). and gives a polynomial of degree $O(d_r)$ and bitsize $\widetilde{O}(d_r^2 + d_r \tau_r)$. It remains to compute the remainder of the division of this polynomial with $f_{J,a}$, which can be done in $\widetilde{O}_B(d_r^4 + d_r^3 \tau_r)$ according to Theorem 2.3.2. A similar computation gives the polynomial $f_{I,a,Y}$, hence the computation of $RUR_{J,a}$ can be done in $\widetilde{O}_B(d_r^4 + d_r^3 \tau_r)$ bit operations. □

Finally, applying the above theorems to the RURs of the ideal $\langle P, Q\rangle$ with P and Q of total degree bounded by d and bitsize bounded by τ yields a complexity in $\widetilde{O}_B(d^8 + d^7\tau)$, for the two previous operations.

6.4 Topology of plane curves

We now address the problem of computing the topology of a real plane algebraic curve $\mathcal{C}_f$ defined by a bivariate polynomial f in $\mathbb{Q}[x, y]$ of total degree d and maximum bitsize τ. As mentioned in the introduction of this thesis, the idea is to follow the approach in [Cheng 2010] and to show how, using the results obtained so far, we can improve their algorithm called ISOTOP.

Before presenting our main contributions, we give some definitions and properties related to real plane algebraic curves and briefly recall the algorithm ISOTOP for computing the topology of curves.

Definition 6.4.1 (**x-critical point**). *A point p is called an x-critical point of an algebraic curve f if* $f(p) = \frac{\partial f}{\partial y}(p) = 0$.

Geometrically the critical points of the curve f are the points of f where the normal vector $(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$ is parallel to the x-axis or vanishes. That means that the tangent to the curve f is either vertical, or not defined. Accordingly, the critical points are distinguished in x-extreme points and singular points.

Definition 6.4.2 (**x-extreme point**). *A point p is called an x-extreme point of a curve f if* $f(p) = \frac{\partial f}{\partial y}(p) = 0$ *and* $\frac{\partial f}{\partial x}(p) \neq 0$.

Definition 6.4.3 (**singular point**). *A point p is called a singular point of a curve f if* $f(p) = \frac{\partial f}{\partial x}(p) = \frac{\partial f}{\partial y}(p) = 0$. *Otherwise p is called regular.*

The following lemma is due to Teissier [Teissier 1973]. It relates the multiplicity of a point $p = (\alpha, \beta)$ in its fiber with respect to f, denoted as $mult(f(\alpha, y), \beta)$, with its multiplicities as an intersection point of the curves $(f, \frac{\partial f}{\partial y})$ and $(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y})$. These intersection multiplicities denoted as $Int(f, \frac{\partial f}{\partial y}, p)$ and $Int(\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}, p)$ are the multiplicities of $p = (\alpha, \beta)$ in the ideals $\langle f, \frac{\partial f}{\partial y}\rangle$ and $\langle \frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\rangle$ as introduced in Definition 5.2.1.

Lemma 6.4.4. *For an x-critical point $p = (\alpha, \beta)$ of the curve f,*

$$mult(f(\alpha, y), \beta) = Int(f, \tfrac{\partial f}{\partial y}, p) - Int(\tfrac{\partial f}{\partial x}, \tfrac{\partial f}{\partial y}, p) + 1.$$

A simpler result can be obtained for the case of x-extreme points, which are known not to cancel the polynomial $\frac{\partial f}{\partial x}$.

Corollary 6.4.5. *For an x-extreme point $p = (\alpha, \beta)$ of f,*

$$mult(f(\alpha, y), \beta) = Int(f, \tfrac{\partial f}{\partial y}, p) + 1.$$

In the sequel, for simplicity, $\frac{\partial f}{\partial y}$ and $\frac{\partial f}{\partial x}$ are denoted respectively, f_y and f_x, we also denote by f_{y^i}, the i-th derivative of f with respect to y. Finally, the multiplicities of critical points in their fibers are all with respect to f.

6.4.1 Isotop outline

In few words, Isotop first focuses on the isolation of the critical points of the curve f, together with the computation of their multiplicity in fiber. Then, using a sweep-line method, the algorithm computes a rectangular decomposition of the plane induced by the boxes of the critical points. Finally, the graph isotopic to f is computed in all rectangles with a greedy method using the multiplicities in fibers. Note that this connection step requires the knowledge of the type of the critical point inside the considered box (singular or x-extreme).

As our contributions solely concern the first step of this algorithm, that is, the isolation of the critical points and the computation of their multiplicities in fiber, we provide below, a more detailed explanation of this step.

Isolation of the critical points. The algorithm first splits the system of critical points I_c, into the system of singular points and the system of extreme points. The system of singular points is $I_s = \{f, f_y, f_x\}$. That of extreme points, denoted I_e, is computed by saturation. More precisely, the equation $1 - uf_x = 0$ is added to the system of critical points, and a Gröbner basis computation is performed to eliminate the variable u and obtain the ideal I_e. The isolation of the solutions of I_s and I_e is done by computing Rational Univariate Representations using the classical strategy that consists in, first, computing Gröbner bases of I_s and I_e, and then computing the corresponding RURs.

Multiplicities of critical points in fibers. For extreme points, the algorithm use the Teissier formula: the multiplicity of an extreme point in I_c is the same as 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 Corollary 6.4.5.

For singular points, since the multiplicity of a point (α, β) in its fiber is, by definition (see Definition 5.2.1) the multiplicity of β in the univariate polynomial $f(\alpha, y)$, this multiplicity is the smallest integer k such that the k-th derivative of $f(\alpha, y)$ does not vanish at β. Accordingly, Isotop solves successively systems $I_{s,k}$, that consist of the system of singular points to which the equations $f_{y^i} = 0$ for $i = 2, \ldots, k$ have been added. At each step, a singular point that is no longer solution of $I_{s,k}$ has its multiplicity in fiber equal to k.

We present in the following, some improvements to Isotop induced by the results of the previous chapter.

6.4.2 Improvements to Isotop

Our first contribution is to change the Gröbner basis and RUR black box in Isotop by the Las-Vegas algorithm of Chapter 5 that computes a decomposition into RURs. More precisely, we consider the system of the critical points defined as $I_c = \{f, f_y\}$ and isolate its real solutions using the algorithm of Chapter 5 (Theorem 5.4.13) for computing the RURs, and the algorithm of Section 6.1 for the isolation step. Solving the system of critical points using these algorithms improves the efficiency of Isotop because, if the system of critical points decomposes, then computing a RUR of one system as in Isotop is much more time consuming than computing RURs of several smaller subsystems, otherwise, if the system does not decompose, the computation of the Gröbner basis tends to be time dominant in Isotop and time consuming compared to computing the triangular systems using subresultants.

On the other hand, we know by Lemma 5.3.1 and Definition 2.2.1, that the multiplicities of the critical points in the RURs computed by the algorithm of Chapter 5 are exactly the multiplicities of these points in their fibers with respect to f. This property is critical for the efficiency of our new algorithm for computing the topology, since it allows to prevent the extra cost of solving several bivariate systems, consisting of f and its successive partial derivatives with respect to y as explained above.

After we have computed the critical points of f with the corresponding multiplicities in fibers, we still need (for the connection step) to distinguish between singular and x-extreme points of f. Here, instead of solving the systems $I_s = \{f, f_y, f_x\}$ and $I_e = \{f, f_y, 1 - uf_x\}$, we use the algorithm of Section 6.3 to split the RURs of the system $\{f, f_y\}$ into two sets of RURs that correspond respectively to the singular points and the x-extreme points of f. According to Theorem 6.3.2, this can be done in complexity $\widetilde{O}_B(d^8 + d^7\tau)$.

We also provide an implementation of this new algorithm called Isotop2, and compared it with state-of-the-art implementations for computing the topology of curves. Corresponding benchmarks are given in Chapter 7.

Finally, one interesting question is, how, or whether, the results we obtained in this thesis may impact the theoretical complexity of CAD-based algorithms for computing the topology. Indeed, most of these algorithms require (sometimes implicitly) the computation of the multiplicities of the critical points in their fiber. Obtaining these multiplicities as a byproduct of solving the critical system should certainly improve the theoretical complexity of some of these algorithms. Similarly, many CAD-based algorithms proceed by shearing the curve in order to put it in generic position. Although very close, this problem is not equivalent to the problem of computing a separating form for the critical points of a curve and a question is to know, if finding a generic shear can be done in the same complexity as computing a separating form.

CHAPTER 7

Implementations and experiments

Contents

We presented in the previous chapters, algorithms for solving bivariate systems and computing with their solutions using Rational Univariate Representations. The complexity analysis of these algorithms shows their theoretical efficiency with resulting complexity bounds that are currently the best available. However, even if the theoretical analysis gives some hints on the relevance of our approach, ultimately, it is above all the practical efficiency that can validate or not this approach.

In order to verify the practical efficiency of our approach, we have implemented the algorithms presented in Chapters 5 and 6 and performed several tests and comparisons with state-of-the-art implementations. We discuss in the next section our implementation work, and then present in Section 7.2 some experimental results.

7.1 Implementations

An algorithm for isolating the real solutions of bivariate systems that follows the approach of Chapter 5 and Section 6.1 has been implemented by Fabrice Rouillier in the RS3 library. RS3 is an open source library written in C that improves the well-known Real solving (RS) library, for the isolation of the real roots of zero-dimensional bivariate algebraic systems.[1] This library provides

---

[1] `https://who.rocq.inria.fr/Fabrice.Rouillier/Home_Page/Software.html`, see also `http://vegas.loria.fr/rs`

the same interface as RS for computations with univariate polynomials. Moreover, since RS3 is based on the algorithm of Chapter 5 for the computation of the RURs, it does not require any prior Gröbner basis computation and thus, does not need a Gröbner basis engine, to the contrary of RS.

The development of the RS3 bivariate solver has been done in parallel to the theoretical work in Chapter 5. Also, several choices we made in our theoretical description are results of practical observations. Despite this, the implementation of the bivariate solver in RS3 slightly deviates from the description given in Chapter 5 in several places. This is because some practical choices in the implementation may substantially improve the practical running time of the algorithm but make its analysis much harder without improving the overall complexity bounds. Moreover, our aim in Chapter 5 was to present in a comprehensive way the essence of our approach, delaying optimization techniques to the implementation phase. An example of that is the assumption we made on the absence of common vertical asymptotes which can artificially pollute the overall complexity of the algorithm. In Section 7.1.1, we point out the main differences between our theoretical description in Chapter 5 and the implementation made by Rouillier.

Another part of our work was the implementation of our algorithms in a new package in the Computational Geometry Algorithms library (CGAL) [cga]. More precisely, we worked on the development of a new model for the bivariate algebraic kernel concept in CGAL, based on the RS3 library. This model gathers necessary tools for solving and handling bivariate polynomial systems (see Section 7.1.2). Since CGAL is written in C++, we created an interface with the RS3 library to benefit from many operations provided by the latter, as for instance, the isolation of the real solutions of bivariate systems or the refinement of the isolating boxes of the solutions. The implementation of this model is described in detail in Section 7.1.2. Note also that for the time being, the package is under prototypical status and very few tests have been achieved. Once the test phase complete this package should be submitted for integration in CGAL.

Finally, we implemented ISOTOP2, a new version of ISOTOP based on our new RS3 solver as described in Section 6.4. We choose to not detail this part of the implementation below, since the initial implementation, ISOTOPhas not suffered from any changes except the replacement of the old RS solver by the new RS3 solver.

7.1.1 RS3 bivariate solver

As mentioned above, aiming at practical efficiency, the implementation of our bivariate solver presents several changes with respect to the description given in Chapter 5. We discuss below these main changes.

The choice of the linear form. It is obvious that choosing a linear form of the form $X+aY$ with $a \neq 0$, instead of the variable X or Y does not change the complexity bound expressed in $\widetilde{O}$ for the RUR-candidates computation. However, one can observe in practice that computing a RUR-candidate using, as a linear form, the variable X or Y leads to intermediate coefficients that are much smaller than those occurring in the computation of a RUR-candidate with respect to a random linear form $X + aY$ with $a \neq 0$. Hence, choosing as a linear form the variable X or Y will obviously speedup the computation of the RUR-candidates. In addition, choosing a random linear form $X + aY$ tends to destroy the sparseness of the system, which results in more symbolic computations impacting negatively the running time of the algorithm.

In our implementation, based on the previous observation, we choose the linear forms starting first with the variables X, Y and then $X + iY$ with $i = 1, \dots, d^4$. In practice however, even if a random choice of the form $X + aY$ is separating with high probability, a mistake would be to consider that this is also the case when it comes to the variable X or Y (because the system modeling a practical problem inherits from its particular structure or symmetries in the original coordinate system). Also, we want to avoid computing the RUR-candidates over $\mathbb{Z}$ with respect to X or Y, and just at the end realizing that the chosen variable is not separating. To avoid the cost of computing completely the RUR-candidates, we design a filter that is able, for a low cost, to exclude a linear form that is non-separating with high probability. More precisely, the idea is to first choose a random linear form $X + aY$ and to compute the corresponding RUR-candidate modulo one arbitrary prime number μ. Then the same computation is performed, but this time with X or Y instead of $X+aY$ and the degrees of the squarefree parts of the resulting RUR-candidates first polynomials are compared. If the degrees are the same and since $X + aY$ is separating with high probability, it is then also the case for the variable X and it can thus be used to compute the RURs over $\mathbb{Z}$. Compared to computing the RUR-candidate associated to one of the variable X or Y over $\mathbb{Z}$, the previous filter is much less expensive, since it amounts to compute only two RUR-candidates modulo a prime number μ.

The choice of the prime numbers. In Algorithm 10, to ease the complexity analysis, the prime numbers for computing the RUR-candidates are chosen randomly in a given set of primes whose cardinal depends on the degree and the bitsize of the input polynomials. This way of choosing primes supposes the prior construction of this set, which turns out to be time consuming in practice, especially when we deal with polynomials with high degree and bitsize. In addition, the primes selected this way have size that also depends on the degree and the bitsize of the input polynomials even though it does not appear in the complexity expression (hidden in the $\widetilde{O}$).

In our implementation, the prime numbers are chosen smaller than the word size of the processor (2^{64}) and thus have bitsize that is independent of the degree and the bitsize of the input polynomial. In addition, unlike in Algorithm 10 where the prime numbers are chosen randomly in a given set, here we avoid constructing this set and select successively primes, starting from 2^{64} and using the routine Prevprime.[2] The rationale is that in practice, the probability that a lot of unlucky prime numbers are located close to 2^{64} is low.

Lifting the result. Another change concerns the way the RURs are lifted over the integers. In our theoretical description, we first compute an upper-bound on the size of these RURs and run the algorithm modulo a set of primes whose product is larger than this bound. However, this bound is not expected to be tight and can be in some cases quite far from the actual size of the RURs, which generally results in unnecessary computations that slow down the algorithm in practice. In our implementation, we avoid the computation of such a theoretical bound and proceed as follows: We select iteratively prime numbers as described above[3], and for each selected prime number μ, we compute the sequence of the RUR-candidates modulo μ, we add this sequence to the set of the already computed sequences $A_{\mathcal{R}}$ and we lift the result over $\mathbb{Z}$ using the sequences in $A_{\mathcal{R}}$. We stop the loop, when two successive lifts yield the same result over $\mathbb{Z}$. Then, with high probability, the computed sequence of RUR-candidates are correct.

Verification. The theoretical description of the verification in Section 5.4.4.1 was guided more by the aim of achieving a good complexity bound than achieving practical efficiency. In our implementation, we keep the same general idea that consists in substituting the coordinates of the RURs in the input polynomials, but we slightly modify the way we perform this substitution. Indeed, instead of evaluating the input polynomials using the monomial basis representation, we use the Horner evaluation scheme, which is known to be more efficient. On the other hand, in the algorithm of Section 5.4.4.1, we first compute the polynomial resulting from the substitution and only after, we compute the reduction of this polynomial by the first polynomial of the RUR, $f_{I,a}$. The drawback of this strategy in practice is that the degree increases at each step of the substitution, which increases the overall cost of the substitution. In our implementation we proceed by reducing modulo the polynomial

---

[2] Prevprime(a) returns the greatest prime number less than or equal to a.

[3] Instead of selecting the prime numbers by sets of m primes, the latter are now selected one by one.

of the RUR $f_{I,a}$, at each step of the substitution so that the intermediate degrees remain low (smaller than the degree of $f_{I,a}$). From the complexity point of view, proceeding this way will considerably increase the bitsize of the intermediate coefficients. In practice however, we observed (without being able to explain it) that the intermediate coefficients do not grow more than the intermediate coefficients in the theoretical description, that is, without reductions.

7.1.2 CGAL Bivariate Algebraic Kernel

The Computational Geometry Algorithms Library is the state of the art library for computational geometry algorithms. Written in C++, this library follows the *generic programming paradigm*, which allows to exchange the types or the methods used by an algorithm. Typically, an algorithm in CGAL can be parametrized by any type as long as the latter satisfies a certain set of requirements called *concept*. Any data type that fulfills the requirement of a concept is called a *model* of this concept.

In CGAL, Algorithms are usually parametrized by a TRAITS CLASS (or a KERNEL) that encapsulates the geometric objects, predicates and constructions used by the algorithm. A typical example is a geometric algorithm, whose traits class defines the input objects the algorithm is working on, and the geometric primitives that are needed to compute with these objects.

While in the past, the focus of the library was on linear objects. Recently, CGAL developers have decided to direct more of the project's attention to non-linear objects. As mentioned in the introduction, handling non-linear objects often requires solving algebraic systems of equations and computing with their solutions. To this purpose, the concept of algebraic kernel has been introduced [Emiris 2008], with the aim of providing a set of functions for computing with polynomials and polynomial systems. Currently, CGAL proposes two different concepts: A univariate algebraic kernel (ALGEBRAICKERNEL_1) that encapsulates basic functionalities for univariate polynomials and algebraic numbers resulting from the isolation of these polynomials, and a bivariate algebraic kernel (ALGEBRAICKERNEL_2) that gathers functions for handling polynomials in two variables.

Currently, two models of the ALGEBRAICKERNEL_1 concept exist in CGAL. The first one is based on an algorithm for isolating the real roots of a univariate polynomial using the bitstream Descartes method [Mehlhorn 2008]. The second one, developed by Peñaranda et al. [Lazard 2009], uses the RS library to perform the root isolation of univariate polynomials [Rouillier 2003].

For the ALGEBRAICKERNEL_2 concept, only one model currently exists. This model is based on algorithms computing a geometric and topological analysis of a single curve [Eigenwillig 2007] and a pair of curves [Eigenwilling 2008]. Roughly, these algorithms follow a Cylindrical-Algebraic-Decomposition approach by first computing the x-coordinate of the critical points of curves and pairs of curves by projection (resultant computation), and then by computing additional information about the fibers of these x-coordinates, for instance, the corresponding y-coordinates, using polynomial subresultants and Sturm-Habicht sequences [Gonzalez-Vega 1989].

Our contribution, which can be seen as an extension of the work of Peñaranda et al. [Lazard 2009], is to propose a new class ALGEBRAICKERNEL_RS_GMPZ_2, which is a model for the ALGEBRAICKERNEL_2 concept, based on the new RS3 bivariate solver. The implementation of this class is an important step toward the integration in CGAL of our algorithm for computing the topology of curves discussed in Section 6.4.

Before presenting the details of our implementation, we review below the main requirements of the ALGEBRAICKERNEL_2 concept. Note that we focus here on the main functionalities and refer the reader to the CGAL manual [Berberich 2013] for a complete description of this concept.

AlgebraicKernel_2 concept. A model for the ALGEBRAICKERNEL_2 concept has to provide the following types and functors:[4]

- COEFFICIENT: A number type that represents the scalar coefficients of bivariate polynomials.
- POLYNOMIAL_2: A type representing bivariate polynomials over the scalar type COEFFICIENT.
- ALGEBRAIC_REAL_2: A type that represents a point in the plane or equivalently a solution of a bivariate system.
- SOLVE_2: A functor that computes the solutions of a bivariate system $S = \{P, Q\}$ of type ALGEBRAIC_REAL_2.
- ISCOPRIME_2, ISSQUAREFREE_2, MAKECOPRIME_2, MAKESQUAREFREE_2, SQUAREFREEFACTORIZE_2: Functors checking for coprimality or squarefreeness, decomposing two polynomials into a common part, and two coprime polynomials, computing the squarefree part and the squarefree factorization.
- COMPUTE_POLYNOMIAL_X_2, COMPUTE_POLYNOMIAL_Y_2, COMPUTEX_2, COMPUTEY_2: Functors computing algebraic num-

---

[4] A functor is an object that defines the operator(). Roughly speaking, it is an object that acts like a function.

ber corresponding to the x and y-coordinates of a solution of a bivariate system as well as the corresponding univariate polynomials.

- IsZeroAt_2, SignAt_2, IsolateX_2, IsolateY_2, Isolate_2: Functors checking the vanishing of a bivariate polynomial at a solution of a bivariate system, computing its sign, computing an isolating box of a solution with respect to the roots of a polynomial or a system of two polynomials.

Note finally that a model of the AlgebraicKernel_2 concept must have as a template argument a model of AlgebraicKernel_1 that provides basic operations for univariate polynomials needed for computations with bivariate polynomials. This model also determines the type of the scalar coefficients of the polynomials.

7.1.2.1 Data representation

Coefficients: Our bivariate algebraic kernel handles polynomials with integer coefficients of arbitrary-length represented by the CGAL type Gmpz. This type is based on the GMP library [Torbjörn Granlund 2002].

Polynomial_2: For representing univariate and bivariate polynomials, we use the CGAL public package for polynomials with arbitrary coefficient types [Hemmer 2010]. This package provides types and operations to deal with multivariate polynomials. In particular, we use operations from this package to implement the functors IsCoprime_2, IsSquareFree_2, MakeCoprime_2, MakeSquareFree_2 and SquareFreeFactorize_2 in our algebraicKernel_2 class.

Algebraic_real_2: In order to fit the output of the RS3 solver, we represent a solution of a bivariate system using the Rational Univariate Representation. More precisely, we represent a solution (α, β) by a data structure that contains the two polynomials of the input system P and Q, the four polynomials of the RUR $\{f(T), f_1(T), f_X(T), f_Y(T)\}$ that encodes the solution, and three intervals corresponding respectively to an isolating interval of $f(T)$, J_γ, and the two intervals $J_\alpha = \frac{\Box f_X(J_\gamma)}{\Box f_1(J_\gamma)}$ and $J_\beta = \frac{\Box f_Y(J_\gamma)}{\Box f_1(J_\gamma)}$ that correspond to the X and Y-coordinates of the solution. Also, in order to ease several operations, we store in this data structure some other relevant information, which we list below.

- An integer m corresponding to the multiplicity of the solution in the corresponding RUR. As shown in Section 6.4, this information is critical for computing the topology of curves using the Isotop approach.

- The integer a of the separating linear form $X + aY$ used to compute the Rational Univariate Representation.

- Two polynomials R_Y and R_X, that vanish respectively at the X-coordinate and the Y-coordinate of the solution. These polynomials are useful to perform operations that do not really fit our Rational Univariate Representation such that COMPUTE_POLYNOMIAL_X_2, COMPUTE_POLYNOMIAL_Y_2, COMPUTEX_2 or COMPUTEY_2 (see below for details). It should be stressed however, that such polynomials are not part of the RS3 solver output and computing them is in general time consuming since it requires the computation of resultants of P and Q. Also, in order to avoid the cost of computing them systematically, the latter are by default initialized to null and computed only on demand, that is, when one of the previous operations is called.

Finally, for interval representations, we use the CGAL number types Gmpfr and Gmpfi, based respectively on the libraries MPFR [MPFR] and MPFI [MPFI]. These number types have been implemented by Luis Peñaranda during the development of the RS-based Univariate algebraic kernel [Lazard 2009] and are integrated as standard number types in CGAL since the version 3.6.

7.1.2.2 Functors implementation

SOLVE_2. Isolating the real solutions of a bivariate system $S = \{P, Q\}$ is the main functionality of the bivariate algebraic kernel. To perform this operation, we developed an interface with the RS3 library. This interface provides two functions that allows respectively to compute the RURs of S and to isolate the solutions of a given RUR.

ISCOPRIME_2, ISSQUAREFREE_2, MAKECOPRIME_2, MAKESQUAREFREE_2, SQUAREFREEFACTORIZE_2. As mentioned previously, all these operations are realized using the basic operations provided by the CGAL package for polynomials.

COMPUTE_POLYNOMIAL_X_2, COMPUTE_POLYNOMIAL_Y_2, COMPUTEX_2, COMPUTEY_2. These operations require the computation of the univariate polynomials R_X and R_Y that vanish respectively at the X and the Y-coordinate of the solution, since the latter are not part of the output of our RS3 solver. This is done by computing the squarefree part of the resultant of the polynomials P and Q (stored in the data structure ALGEBRAIC_REAL_2) with respect to X and Y. The two operations COMPUTEX_2, COMPUTEY_2 require in addition the computation of isolating intervals of R_X (resp. R_Y)

that correspond to the X-coordinate (resp. the Y-coordinate) of the solution. This is done by isolating the roots of the univariate polynomial R_X (resp. R_Y) and then, refining the isolating box of the solution until the interval corresponding to the X-coordinate (resp. Y-coordinate) overlaps only one isolating interval of the univariate polynomials R_X (resp. R_Y).

For refining the isolating box of a solution, we use the quadratic refinement method provided by the RS3 interface. This method, which takes a RUR as input, first refines the isolating interval of f corresponding to the solution and then compute the intervals corresponding to its X and Y-coordinate by evaluating the rational fractions of the RUR using interval arithmetic.

IsZeroAt_2, SignAt_2, IsolateX_2, IsolateY_2, Isolate_2. The implementations of these functors are all based on the same basic idea. It consists in checking whether the considered solution is solution of the given polynomials. If so, we have finished. Otherwise, we refine the box of the solution and evaluate the polynomials at this box using interval arithmetic, until 0 is not contained in the resulting interval so that we can conclude.

For checking that a solution (α, β) is a zero of a given polynomial $F(X, Y)$, we proceed as in Section 6.2 by first computing the polynomial f_F resulting from the substitution of the RUR coordinates associated to (α, β) in the polynomial $F(X, Y)$. Then we compute the squarefree part of the gcd of f_F and f and compute its signs at the two bounds of the isolating interval of f. If the obtained signs are different, this means that the solution (α, β) is also a solution of $F(X, Y)$.

7.2 Experiments

Our tests consist of two parts: In the first part, we test the RS3 solver for the isolation of the real solutions of bivariate systems and in the second part we test isotop2 for the computation of the topology of curves.

The test suite includes 1500 examples of curves of degree up to 100 that are already used in [Labs 2009, Brown 2002, Gonzalez-Vega 2002, Berberich 2011a, Berberich 2011b, Cheng 2010]. For the bivariate solving tests, we consider the systems defined by the curves and their derivatives with respect to the variable y. In the following, we only keep instances for which the running time of RS3 is larger than 1 second on a regular laptop. This reduces our data base to about 300 examples. We also consider one challenging example from [Cazals 2008]. The timeout is set to 60 minutes for the RS3 experiments and to 30 minutes for the Isotop2 experiments.

7.2.1 RS3 experiments

We run our experiments on 2.27GHz 6-Core Intel Xeon L5640 with 8MB of L2 cache under Linux platform, and compare our RS3 solver in its sequential and multi-thread versions (with respectively three and six threads), with the bivariate version of the solver of the Maple package Regular Chains [Li 2011], which we refer to as IsolateRC, and Lgp by Jin-San Cheng et al. [Cheng 2009] (both implemented in maple).[5] We measure separately the running times of the two steps of our RS3 solver, that is, the symbolic step that consists in computing the RURs and the numerical step in which isolating boxes for the solutions are computed from these RURs, and we add as markers the running times of computing the resultant on Maple and those of performing the triangular decomposition (Triangularize) and the radical triangular decomposition (Radical_Triangularize) in the Regular Chains Solver. This allows first to compare the cost of computing the RURs to that of computing only the resultant or the triangular decomposition, and second to measure the gain that the RURs computation actually brings to the numerical step. In addition, since only the symbolic part of our RS3 solver is multi-thread, isolating its running time allows to measure precisely the impact of the use of multi-thread implementations in the global running time, especially for instances where the symbolic part is known to be the bottleneck of the solving process.

We study the performance of our RS3 solver on different classes of examples. We first consider a set of "special" curves taken from [Berberich 2011b]. These curves have, for instance, a large number of solutions, high-curvature points, many solutions on the same vertical line or many clustered solutions. The timings and the descriptions of these curves are given in Table 7.1 and Table 7.2. Second, we consider, a set of home-made instances. These instances include generic curves, generated either as random bivariate polynomials, or as resultants of two random trivariate polynomials, and non-generic curves, generated as the product of random curves with one or several of their vertical translates so that each fiber will contain more than one critical point.[6] In each case, the curves are considered with increasing total degrees and coefficient bitsizes. The timings for these curves are given in Tables 7.3, 7.4 and 7.5.

A main observation that stems from our experiments is that RS3, in its sequential version, is generally faster than the two contestants. For IsolateRC,

---

[5] Note that we do not compare our solver to the recent solver Bisolve [Berberich 2011b] since in the next section, we compare our ISOTOP2 implementation for computing the topology of curves with FastAnalysis implementation from [Berberich 2011a], which is shown to be quite uniformly faster than Bisolve.

[6] Each curve $f(x, y)$ is multiplied by one or several curves of the form $f(x, y + l)$ with $l = 1, 2, 3, \ldots$.

the gain factor is, on average, equal to 140 varying between 0.4 and 1000 while for Lgp, it is, on average, equal to 5 and varies between 0.1 and 40.

In addition, by comparing columns 2, 7 and 8 in our tables, one can see that first, RC Triangularize is very efficient since it is only slower than the resultant of Maple by a factor less than 10 although Triangularize computes a lot more than the resultant, and second, that our computation of the RURs (including the decomposition into triangular subsystems) is comparable to Triangularize in the sense that the time ratio varies between 0.1 and 10.

On the other hand, comparing columns 3 and 10, which correspond to the running times for only the isolation steps, we observe that first, the running time of this step in RS3 is, except for very few instances, negligible compared to the running time for computing the RURs and second, that computing isolating boxes from the RURs is much more efficient than computing them from regular triangular systems. Moreover, unlike RS3 where the isolation step follows directly after the RURs computation, the solver IsolateRC requires first the computation of the radical triangular decomposition before performing the isolation of the solutions. Computing such a radical decomposition may induce a substantial overhead compared to only computing the triangular decomposition, especially for non-generic curves (see column 9 in Table 7.5).

We thus conclude from the above observations that the overhead of the symbolic computation of the RURs (from the triangular systems) is small compared to the significant benefit it yields for the isolation step.

Finally, we observe that the use of multi-thread implementations brings a notable gain to the running times of the symbolic step of RS3, which is expected since we use multi-modular arithmetic. One can however notice that the gained factor is more important between the sequential version and the multi-thread version with three threads (a factor between 2 and 3), than between the latter and the multi-thread version with six threads especially for curves with large coefficients (see Table 7.1). So far, we are not able to explain this fact. Note finally that in most cases, the gained factor resulting from the use of the multi-thread implementation is reflected in the running time of the global RS3 solver; this is because the isolation step is in general negligible compared to the RURs computation step.

Curves	RURs_Seq	Isolation	**RS3_Seq**	RURs_MT3	RURs_MT6	Resultant	Triangularize	Tr_radical	RC_Iso_step	**IsolateRC**	**LGP**
13_sings_9	0.17	0.08	**0.25**	0.07	0.07	0.55	3.03	2.79	13.75	**16.54**	**9.30**
FTT_5_4_4	9.524	5.256	**14.78**	2.17	1.19	0.97	0.75	1.00	10.27	**11.27**	**216.71**
SA_2_4_eps	0.386	0.284	**0.67**	0.05	0.02	1.38	5.66	5.68	0.17	**5.85**	**5.13**
SA_4_4_eps	10.74	0.84	**11.58**	4.44	3.46	6.00	90.50	90.39	1.08	**91.47**	**152.89**
compact_surf	2.916	1.244	**4.16**	0.91	0.58	0.60	11.93	13.16	156.33	**169.49**	**15.01**
cov_sol_20	39.907	12.573	**52.48**	9.49	5.68	1.00	114.36	443.87	Timeout	**Timeout**	**67.69**
curve24	11.334	2.936	**14.27**	4.27	3.34	6.48	9.90	10.10	3083.85	**3093.95**	**45.90**
curve_issac	0.256	0.244	**0.50**	0.05	0.06	0.28	2.03	2.38	464.59	**466.97**	**2.81**
cusps_and_flexes	1.684	0.716	**2.40**	0.96	0.82	0.97	3.47	3.67	129.99	**133.66**	**2.97**
deg16_7_curves	0.348	0.132	**0.48**	0.14	0.13	0.43	7.18	7.92	1.23	**9.15**	**4.71**
degree_6_surf	11.814	1.116	**12.93**	4.52	3.25	7.20	398.25	398.13	Timeout	**Timeout**	**337.64**
mignotte_xy	221.095	12.435	**233.53**	153.32	148.45	51.50	247.23	248.22	Timeout	**Timeout**	**1077.57**
dfold_10_6	0.759	0.221	**0.98**	0.30	0.21	0.05	0.55	0.55	41.95	**42.50**	**5.43**
grid_deg_10	8.329	0.861	**9.19**	5.03	4.63	1.31	4.36	4.76	63.42	**68.18**	**2.80**
huge_cusp	116.254	1.136	**117.39**	112.95	112.96	8.15	35.29	35.90	676.05	**711.95**	**140.30**
rand_9_2048	81.472	0.328	**81.80**	72.46	71.59	3.81	17.95	17.67	16.02	**33.69**	**7.41**
spider	79.341	16.199	**95.54**	26.26	19.49	33.83	281.29	287.22	Timeout	**Timeout**	**Timeout**
swinnerson	1.534	0.236	**1.77**	0.73	0.54	0.62	13.93	14.20	449.82	**464.02**	**35.74**
ten_circles	0.298	0.182	**0.48**	0.09	0.09	0.36	7.37	7.86	1.30	**9.16**	**4.62**

Table 7.1: Benchmark results for the “special” curves listed in Table 7.2. Running times are in seconds

Instance	**Description**	**Instance**	**Description**
13_sings_9	large coefficients, high-curvature points	FTT_5_4_4	many non-rational singularities
SA_2_4_eps	singular points with high tangencies, displaced	SA_4_4_eps	singular points with high tangencies, displaced
compact_surf	silhouette of an algebraic surface, many singularities, isolated solutions	cov_sol_20	covertical solutions
curve24	curvature of degree 8 curve, many singularities	curve_issac	isolated points, high-curvature points
cusps_and_flexes	high-curvature points	deg16_7_curves	a set of 7 random curves multiplied together
degree_6_surf	silhouette of an algebraic surface, covertical solutions in both directions	mignotte_xy	a product of $x/y-$ Mignotte polynomials, displaced, many clustered solutions
dfold_10_6	a curve with many half-branches	grid_deg_10	large coefficients, curve in generic position
huge_cusp	large coefficients, high-curvature points	rand_9_2048	random curve with large coefficients
spider	degenerate curve, many clustered solutions	swinnerson	covertical solutions in both directions
ten_circles	a set of 10 random circles multiplied together		

Table 7.2: Description of the curves whose running times are given in Table 7.1. The polynomials defining these curves are archived online at `http://www.mpi-inf.mpg.de/departments/d1/projects/Geometry/BisolveDatasetAlenex2011`.

d, τ, density	RURs_Seq	Isolation	**RS_Seq**	RURs_MT3	RURs_MT6	Resultant	Triangularize	Triangularize_radical	RC_Iso_step	**IsolateRC**	**LGP**
10,32	0.148	0.332	**0.48**	0.12	0.07	0.37	0.48	0.45	17.07	**17.52**	**0.68**
12,32	0.552	0.308	**0.86**	0.13	0.07	0.40	0.70	0.69	22.23	**22.92**	**0.74**
14,32	1.156	0.384	**1.54**	0.43	0.35	0.52	0.90	0.91	39.08	**39.99**	**1.71**
16,32	2.592	0.468	**3.06**	1.07	0.88	0.99	2.64	2.62	52.30	**54.92**	**2.72**
18,32	4.53	0.56	**5.09**	1.98	1.66	1.75	5.62	5.64	60.99	**66.63**	**3.84**
20,32	7.032	1.008	**8.04**	2.92	2.31	3.01	10.88	10.87	126.74	**137.61**	**16.12**
8,1024,sparse	12.31	0.2	**12.51**	10.58	9.58	1.36	2.91	2.91	8.51	**11.42**	**8.08**
16,32,sparse	2.132	0.488	**2.62**	0.852	0.732	0.73	1.57	1.57	60.53	**62.10**	**2.69**
16,1024,sparse	163.196	0.764	**163.96**	120.25	113.93	15.50	202.16	202.31	76.59	**278.90**	**26.09**
32,1024,sparse	2271.056	613.504	**2884.56**	1320.196	1176.016	263.18	Timeout	Timeout	Timeout	**Timeout**	**Timeout**
8,1024,dense	11.258	0.392	**11.65**	8.96	8.70	1.25	2.72	2.77	16.69	**19.46**	**2.28**
16,32,dense	2.812	0.308	**3.12**	1.23	1.02	0.73	2.20	2.19	29.80	**31.99**	**2.26**
16,1024,dense	174.278	0.732	**175.01**	128.73	122.33	16.10	369.21	371.25	54.70	**425.95**	**28.23**
32,32,dense	60.056	7.504	**67.56**	23.71	17.85	27.73	327.34	328.03	1417.33	**1745.36**	**121.43**

Table 7.3: Benchmark results for random curves of degree d and bitsize τ. When the density of the polynomials is not specified, it is equal to 50%. Running times are in seconds

d_1, d_2, τ	RURs_Seq	Isolation	**RS_Seq**	RURs_MT3	RURs_MT6	Resultant	Triangularize	Triangularize_radical	RC_Iso_step	**IsolateRC**	**LGP**
3,3,32	0.23	0.11	**0.34**	0.08	0.06	0.37	1.97	1.81	24.22	**26.03**	**1.17**
4,4,16	1.546	0.524	**2.07**	0.48	0.36	1.41	18.30	18.20	2889.88	**2908.08**	**5.95**
4,4,32	5.686	0.484	**6.17**	2.53	2.12	4.49	196.79	197.85	Timeout	**Timeout**	**16.73**
5,5,16	10.284	1.676	**11.96**	3.71	2.72	7.42	641.23	636.55	Timeout	**Timeout**	**175.79**
5,4,32	2.49	0.91	**3.40**	0.69	0.47	2.18	53.49	53.49	Timeout	**Timeout**	**17.59**
3,3,16,translated	2.926	1.244	**4.17**	0.71	0.38	3.99	77.61	90.88	2041.87	**2132.75**	**16.97**
4,3,16,translated	19.582	1.688	**21.27**	7.55	5.64	20.42	1183.04	1402.25	Timeout	**Timeout**	**122.99**

Table 7.4: Benchmark results for resultants of two random trivariate polynomials of degrees d_1, d_2 and bitsize τ with 50% of non-zero coefficients. In the two last rows, the obtained resultant is translated once. Running times are in seconds.

d, τ, nb_tr	RURs_Seq	Isolation	**RS3_Seq**	RURs_MT3	RURs_MT6	Resultant	Triangularize	Radical	RC_Iso_step	**IsolateRC**	**LGP**
4,32,2	0.318	0.132	**0.45**	0.09	0.06	0.72	2.24	3.16	1.85	**5.01**	**1.47**
4,32,3	0.79	0.42	**1.21**	0.21	0.08	1.62	9.51	24.82	85.64	**110.46**	**6.76**
5,32,2	0.854	0.396	**1.25**	0.17	0.09	1.15	8.21	13.06	10.19	**23.25**	**3.15**
5,32,3	5.134	0.996	**6.13**	1.73	1.21	3.49	47.05	151.75	40.41	**192.16**	**13.50**
6,32,2	3.114	0.996	**4.11**	0.85	0.50	2.08	27.77	53.25	127.56	**180.81**	**10.81**
6,32,3	8.268	2.392	**10.66**	2.07	1.11	5.71	120.05	456.59	Timeout	**Timeout**	**47.32**
7,32,2	8.156	1.384	**9.54**	2.79	2.08	3.76	95.23	228.56	820.42	**1048.98**	**27.63**
7,32,2	8.081	1.149	**9.23**	2.82	2.00	3.47	99.53	179.46	Timeout	**Timeout**	**23.14**
7,32,3	24.146	2.404	**26.55**	8.41	6.07	11.46	475.74	772.45	993.01	**1765.46**	**97.35**
7,32,3	25.544	2.576	**28.12**	8.98	6.61	14.01	39.19	57.20	619.57	**676.77**	**119.39**
8,32,1	2.344	0.596	**2.94**	1.01	0.50	0.95	2.96	3.43	142.40	**145.83**	**6.10**
8,32,2	14.866	1.244	**16.11**	5.49	4.08	5.70	22.78	29.00	714.78	**743.78**	**39.40**
8,32,3	67.43	3.06	**70.49**	25.41	19.41	31.42	2356.67	Timeout	Timeout	**Timeout**	**233.87**

Table 7.5: Benchmark results for random curves of degree d, bitsize τ with 50% of non-zero coefficients which have been translated nb_tr times. Running times are in seconds

7.2.2 ISOTOP2 experiments

We compare our implementation ISOTOP2 with ISOTOP, [Cheng 2010], CA, the arrangement package of CGAL [Eigenwillig 2007] and FastAnalysis [Berberich 2011a]; some comparisons with older algorithms can be found in these papers, in particular in [Cheng 2010].

Isotop2 versus Isotop. Comparing ISOTOP2 with ISOTOP [Cheng 2010], we observe that ISOTOP2 is uniformly more efficient than ISOTOP . We observe a significant ratio of running time. For generic curves, the ratio varies between 5 and 15 on curves of degree between 10 and 20. For non-generic curves, we observe a ratio between 1.5 and 15 for degree between 12 and 25. Naturally, the ratio increases with the degree of the curves. Corresponding benchmarks are given in Tables 7.6, 7.7, 7.8, 7.9 and 7.10.

Isotop2 versus CA (CGAL). Comparing ISOTOP2 with CA [Eigenwillig 2007], the curve analysis of the arrangement package of CGAL, we observe that for small instances or generic ones, the running times are similar with a small advantage for CA (of a factor at most 3). On the other hand, for non-generic curves and particularly for high degree ones, ISOTOP2 outperforms CA by factors ranging from 1 to 500, and on the most difficult instances CA does not terminate in less than 30 minutes (on a regular PC). Corresponding benchmarks are given in Table 7.11.

Isotop2 versus FastAnalysis. The algorithm presented in [Berberich 2011a] and its implementation *FastAnalysis* reduce as much as possible the (practical) cost of the symbolic computations by running a very fast GPU implementation for computing resultants. Note that *FastAnalysis* and ISOTOP2 share the same implementation (RS [Rouillier 2003]) for isolating the roots of univariate polynomials with rational coefficients.

FastAnalysis is not yet distributed and we considered the instances presented in [Berberich 2011a]. We first compared *FastAnalysis* to the sequential version of ISOTOP2 running on a comparable machine as in [Berberich 2011a] and observed comparable behavior on most instances with an advantage to *FastAnalysis* of a factor up to 4. We then compared *FastAnalysis* to the multi-thread version of our algorithm. We observe that the two implementations perform similarly with an advantage to *FastAnalysis* of a factor between 1/2 and 3.[7] Although the multi-thread version of RS3 is essentially linear in the number of threads, this does not substantially change the comparative behavior because the symbolic part of the computation (the RURs) becomes negligible compared to the numeric phase (isolation) in the topology com-

---

[7] At this stage our running times of our multi-thread version of ISOTOP2 are extrapolated because, for technical reasons, only our solver RS3 is multi-thread.

putation part of ISOTOP2, which is sequantial in the current version. The benchmark results are reported in Table 7.12.

Challenging curves. We also ran ISOTOP2 on a challenging curve, called *ridge*, coming from [Cazals 2008]. This curve was so far unreachable without manually driving the computations using computer algebra savoir-faire and some information on the structure of the curve. To our knowledge, this is the first time that this curve can be handled with a black-box algorithm. The polynomial defining *ridge* has degree 84 (43 in each variable), 1907 monomials with coefficients of 53 digits. This curve has 1432 extreme points and 909 singular ones. The sequential version of ISOTOP2 terminates in 2 hours on the same regular laptop as above (while we interrupted CA after three days) and in about 10 mn on a regular laptop with 8 threads.[7]

degree	τ	Isotop	Isotop2	$\frac{\text{Isotop}}{\text{Isotop2}}$
10	32	3.8	0.76	**5**
12	32	9.7	1.49	**6.5**
14	32	22	2.58	**8.5**
16	32	72	7.2	**10**
18	32	160	13.33	**12**
20	32	320	22.85	**14**

Table 7.6: Running times in seconds (averaged over 5 runs) for random bivariate polynomials with 50% non-zero coefficients of bitsize 32.

degree	τ	Isotop	Isotop2	$\frac{\text{Isotop}}{\text{Isotop2}}$
16	64	29	3.81	**7.6**
25	80	590	39.33	**15**

Table 7.7: Running times (averaged over five runs) in seconds for resultants of two random trivariate polynomials, both of total degree 4 or 5, with 50% non-zero coefficients of bitsize 8.

degree	τ	Isotop	Isotop2	$\frac{\text{Isotop}}{\text{Isotop2}}$
12	96	4.1	2.73	**1.5**
15	96	15	5	**3**
18	96	49	14	**3.5**
21	96	140	28	**5**

Table 7.8: Running times (averaged over five runs) in seconds for non-generic curves generated by the product of a curve f with two of its translates $f(x, y+1)$ and $f(x, y+2)$. The curve f is chosen randomly with degree between 4 and 7 and 50% non-zero coefficients of bitsize 32.

d_1	d_2	degree	τ	Isotop	Isotop2	$\frac{\text{Isotop}}{\text{Isotop2}}$
3	3	18	96	26	7.42	**3.5**
3	4	24	112	250	39.68	**6.3**

Table 7.9: Running times (averaged over five runs) in seconds for non-generic curves generated by the product of a curve f with its translate $f(x, y + 1)$. The curve f is the resultant of two random trivariate polynomials of total degree d_1 and d_2 and 50% non-zero coefficients of bitsize 8.

degree	τ	Isotop	Isotop2	$\frac{\text{Isotop}}{\text{Isotop2}}$
20	64	39	16.95	**2.3**
24	64	240	80	**3**
28	64	350	100	**3.5**

Table 7.10: Running times (averaged over five runs) in seconds for polynomials of the form $g = f^2(x, y) + f^2(x, -y)$. The random polynomials f have bitsize 32, term density 50% and degrees 10, 12, 14.

Table 7.11: Benchmark results of Isotop2 and CA (Curve Analysis) run on a set of curves taken from [Labs 2009, Brown 2002, Gonzalez-Vega 2002, Cheng 2010] whose descriptions can be found in [Peñaranda. 2010]. The second column specifies the degree of the curve, its bitsize, the number of its singular points, the number of its x-extreme points and the number of its vertical asymptotes. We run two versions of CA, CA0 and CA1. The implementation CA1 being optimized for generic curves. CA min corresponds to the best running time between CA0 and CA1. The timeout is set to 30 minutes.

$\mathcal{C}$	d	τ	s	e	a	**Isotop2**	CA0	CA1	**CA min**	$\frac{\text{CA}}{\text{Isotop2}}$
$L_{16,012}$	97	22	4	0	0	**519.04**	Timeout	Timeout	**Timeout**	>3
$L_{16,016}$	97	14	4	0	0	**361.90**	Timeout	Timeout	**Timeout**	>4
$L_{16,021}$	97	10	4	0	0	**345.27**	Timeout	Timeout	**Timeout**	>5
$L_{17,075}$	65	140	2	0	0	**228.46**	630.36	1236.55	**630.36**	2.76
$L_{16,011}$	81	18	4	0	0	**218.55**	Timeout	Timeout	**Timeout**	>8
$L_{17,074}$	65	113	2	0	0	**193.68**	477.82	1010.70	**477.82**	2.47
$L_{16,026}$	81	7	4	0	0	**158.59**	Timeout	Timeout	**Timeout**	>11
$L_{17,073}$	65	86	2	0	0	**150.43**	352.60	779.11	**352.60**	2.34
$L_{17,077}$	65	60	6	0	0	**137.57**	Timeout	Timeout	**Timeout**	>13
$L_{17,072}$	65	60	2	0	0	**121.71**	223.46	578.86	**223.46**	1.84
$L_{2,84}$	100	8	1	0	0	**104.45**	200.38	Timeout	**200.38**	1.92
$L_{16,015}$	73	10	4	0	0	**100.71**	Timeout	Timeout	**Timeout**	>17
$L_{17,076}$	65	33	6	0	0	**100.42**	Timeout	Timeout	**Timeout**	>17
$L_{17,071}$	65	33	2	0	0	**90.91**	121.83	429.16	**121.83**	1.34
$L_{16,010}$	65	14	4	0	0	**75.99**	Timeout	Timeout	**Timeout**	>23
$L_{17,060}$	49	140	6	0	0	**72.86**	Timeout	Timeout	**Timeout**	>24
$L_{16,020}$	65	7	4	0	0	**70.18**	Timeout	Timeout	**Timeout**	>25
$L_{17,055}$	49	140	2	0	0	**67.01**	194.56	338.38	**194.56**	2.90
$L_{17,059}$	49	113	6	0	0	**64.79**	Timeout	Timeout	**Timeout**	>27
$L_{2,79}$	90	8	1	0	0	**64.61**	122.38	1601.18	**122.38**	1.89
$L_{17,054}$	49	113	2	0	0	**55.89**	148.20	266.04	**148.20**	2.65
$L_{17,058}$	49	86	6	0	0	**52.17**	Timeout	Timeout	**Timeout**	>34
$M_{7,3,1}$	28	127	30	16	0	**51.43**	Timeout	Timeout	**Timeout**	>35
$L_{17,053}$	49	86	2	0	0	**46.18**	104.60	202.84	**104.60**	2.27
$L_{2,83}$	80	7	1	0	0	**42.00**	51.54	670.79	**51.54**	1.23
$L_{17,057}$	49	60	6	0	0	**40.11**	Timeout	Timeout	**Timeout**	>44
$L_{2,74}$	80	8	1	0	0	**38.87**	78.48	852.01	**78.48**	2.02
$L_{17,052}$	49	60	2	0	0	**35.22**	65.69	145.56	**65.69**	1.87
$L_{16,031}$	49	3	4	0	0	**34.21**	Timeout	Timeout	**Timeout**	>52
$L_{16,006}$	49	22	4	0	0	**31.97**	Timeout	Timeout	**Timeout**	>56
$L_{17,056}$	49	33	6	0	0	**28.25**	Timeout	Timeout	**Timeout**	>63
$M_{6,3,2}$	24	128	24	12	1	**25.72**	754.43	826.97	**754.43**	29.33

$\mathcal{C}$	d	τ	s	e	a	**Isotop2**	CA0	CA1	**CA min**	$\frac{\text{CA}}{\text{Isotop2}}$
$L_{2,78}$	72	7	1	0	0	**25.23**	32.52	376.05	**32.52**	1.29
$L_{17,051}$	49	33	2	0	0	**24.26**	34.92	98.61	**34.92**	1.44
$M_{6,3,4}$	24	129	20	24	0	**22.84**	767.37	627.94	**627.94**	27.49
$L_{2,69}$	70	8	1	0	0	**20.82**	42.64	423.97	**42.64**	2.05
$M_{6,3,1}$	24	127	6	4	0	**18.26**	415.34	468.27	**415.34**	22.74
$L_{17,070}$	33	70	6	0	0	**18.10**	Timeout	Timeout	**Timeout**	>99
$M_{6,3,5}$	24	130	24	8	1	**17.95**	555.22	546.35	**546.35**	30.44
$L_{16,009}$	49	10	4	0	0	**17.54**	Timeout	Timeout	**Timeout**	>102
$M_{7,2,1}$	21	97	14	12	0	**16.92**	426.24	436.36	**426.24**	25.19
$L_{17,040}$	33	140	6	0	0	**16.83**	Timeout	Timeout	**Timeout**	>106
$L_{16,014}$	49	7	4	0	0	**15.57**	Timeout	Timeout	**Timeout**	>115
$M_{7,2,3}$	21	98	8	12	1	**15.11**	391.74	312.81	**312.81**	20.70
$L_{2,73}$	64	7	1	0	0	**14.67**	20.86	201.53	**20.86**	1.42
$M_{7,2,2}$	21	97	18	18	0	**14.58**	303.73	337.97	**303.73**	20.83
$L_{17,069}$	33	56	6	0	0	**14.28**	Timeout	Timeout	**Timeout**	>126
$L_{17,035}$	33	140	2	0	0	**13.95**	45.97	72.03	**45.97**	3.30
$L_{17,039}$	33	113	6	0	0	**13.44**	Timeout	Timeout	**Timeout**	>133
$L_{16,025}$	41	3	4	0	0	**13.17**	Timeout	Timeout	**Timeout**	>136
$L_{2,82}$	60	5	1	0	0	**12.90**	9.17	106.15	**9.17**	0.71
$L_{16,005}$	41	18	4	0	0	**12.68**	Timeout	Timeout	**Timeout**	>141
$M_{5,3,5}$	20	128	10	8	0	**12.64**	203.85	225.35	**203.85**	16.13
$M_{7,2,4}$	21	96	14	9	0	**12.40**	221.64	269.65	**221.64**	17.87
$L_{17,034}$	33	113	2	0	0	**12.25**	34.66	54.17	**34.66**	2.83
$M_{5,3,1}$	20	130	26	8	1	**12.20**	283.25	257.21	**257.21**	21.08
$L_{17,038}$	33	86	6	0	0	**12.10**	Timeout	Timeout	**Timeout**	>148
$L_{17,068}$	33	43	6	0	0	**11.82**	Timeout	804.64	**804.64**	68.07
$L_{2,64}$	60	8	1	0	0	**10.66**	24.89	183.40	**24.89**	2.33
$L_{17,065}$	33	70	2	0	0	**10.65**	4.70	11.18	**4.70**	0.44
$M_{5,3,4}$	20	129	16	12	0	**10.35**	204.12	190.90	**190.90**	18.44
$L_{13,054}$	32	14	1	20	0	**10.24**	491.27	390.24	**390.24**	38.11
$L_{17,033}$	33	86	2	0	0	**10.11**	23.82	39.81	**23.82**	2.36
$L_{17,067}$	33	30	6	0	0	**9.11**	Timeout	Timeout	**Timeout**	>197
$L_{17,064}$	33	56	2	0	0	**9.07**	3.93	9.56	**3.93**	0.43
$M_{6,2,2}$	18	100	6	12	0	**8.97**	46.40	63.03	**46.40**	5.17
$M_{6,2,3}$	18	98	16	18	1	**8.97**	61.30	62.64	**61.30**	6.83
$L_{17,037}$	33	60	6	0	0	**8.66**	1753.89	Timeout	**1753.89**	202.50
$L_{6,032}$	10	22	0	61	0	**8.39**	39.02	38.93	**38.93**	4.64
$L_{6,031}$	10	18	0	61	0	**8.32**	38.67	38.63	**38.63**	4.64
$M_{5,3,2}$	20	128	18	12	1	**8.25**	178.51	176.11	**176.11**	21.34
$L_{6,029}$	10	12	0	61	0	**8.25**	62.77	62.48	**62.48**	7.57
$L_{6,030}$	10	15	0	61	0	**8.21**	38.15	37.95	**37.95**	4.62
$L_{2,68}$	56	7	1	0	0	**7.93**	12.58	99.52	**12.58**	1.59

$\mathcal{C}$	d	τ	s	e	a	**Isotop2**	CA0	CA1	**CA min**	$\frac{\text{CA}}{\text{Isotop2}}$
$L_{2,77}$	54	5	1	0	0	**7.92**	6.11	61.77	**6.11**	0.77
$M_{5,3,3}$	20	132	18	12	0	**7.92**	110.11	109.87	**109.87**	13.87
$L_{3,35}$	9	75	0	69	0	**7.75**	59.59	59.71	**59.59**	7.69
$L_{17,032}$	33	60	2	0	0	**7.71**	14.84	28.17	**14.84**	1.92
$L_{17,066}$	33	16	6	0	0	**7.49**	Timeout	Timeout	**Timeout**	>240
$L_{13,065}$	34	15	1	24	0	**7.41**	1187.52	Timeout	**1187.52**	160.24
$L_{17,063}$	33	43	2	0	0	**7.41**	3.24	8.34	**3.24**	0.44
$L_{3,34}$	9	59	0	68	0	**7.03**	52.34	51.20	**51.20**	7.28
$L_{17,036}$	33	33	6	0	0	**6.99**	Timeout	Timeout	**Timeout**	>257
$L_{5,216}$	36	18	0	0	0	**6.97**	2.83	7.99	**2.83**	0.41
$M_{6,2,4}$	18	96	9	15	1	**6.91**	48.34	46.83	**46.83**	6.78
$L_{13,064}$	32	14	1	24	0	**6.72**	319.14	280.31	**280.31**	41.71
$L_{3,33}$	9	45	0	67	0	**6.48**	54.05	54.11	**54.05**	8.34
$L_{13,060}$	34	15	1	24	0	**6.21**	526.40	635.12	**526.40**	84.77
$M_{6,2,1}$	18	97	18	18	0	**6.11**	46.80	55.29	**46.80**	7.66
$L_{3,32}$	9	35	0	66	0	**6.10**	49.38	49.35	**49.35**	8.09
$L_{16,019}$	33	3	4	0	0	**6.06**	Timeout	Timeout	**Timeout**	>297
$L_{17,062}$	33	30	2	0	0	**5.88**	2.54	6.97	**2.54**	0.43
$M_{4,3,5}$	16	129	26	16	0	**5.77**	36.97	38.64	**36.97**	6.41
$L_{3,31}$	9	29	0	65	0	**5.70**	95.19	94.73	**94.73**	16.62
$L_{17,031}$	33	33	2	0	0	**5.66**	7.72	16.67	**7.72**	1.36
$L_{13,042}$	28	12	1	16	0	**5.61**	153.28	430.32	**153.28**	27.32
$M_{6,2,5}$	18	99	13	9	1	**5.52**	33.53	37.11	**33.53**	6.07
$L_{17,050}$	25	70	6	0	0	**5.38**	1154.31	1102.13	**1102.13**	204.86
$L_{11,066}$	24	27	12	24	0	**5.35**	71.61	72.27	**71.61**	13.38
$L_{5,215}$	36	15	0	0	0	**5.32**	1.90	6.93	**1.90**	0.36
$L_{11,060}$	24	25	6	12	0	**5.17**	35.88	36.39	**35.88**	6.94
$L_{3,30}$	9	54	0	62	0	**5.13**	17.21	17.26	**17.21**	3.35
$L_{2,59}$	50	8	1	0	0	**5.09**	12.66	73.45	**12.66**	2.49
$L_{5,214}$	36	11	0	0	0	**4.84**	5.10	10.00	**5.10**	1.05
$L_{13,059}$	32	14	1	24	0	**4.82**	647.77	130.74	**130.74**	27.13
$M_{5,2,5}$	15	97	16	18	0	**4.81**	21.25	20.64	**20.64**	4.29
$L_{3,29}$	9	37	0	61	0	**4.74**	14.48	14.39	**14.39**	3.04
$L_{17,049}$	25	56	6	0	0	**4.73**	1248.89	1291.35	**1248.89**	264.04
$L_{11,065}$	24	25	11	22	0	**4.59**	54.87	55.15	**54.87**	11.95
$L_{2,72}$	48	5	1	0	0	**4.57**	3.78	33.17	**3.78**	0.83
$L_{10,064}$	24	39	10	20	0	**4.54**	50.72	50.42	**50.42**	11.11
$L_{10,064}$	24	39	10	20	0	**4.47**	49.73	50.02	**49.73**	11.12
$L_{11,058}$	24	23	4	8	0	**4.46**	23.76	24.26	**23.76**	5.33
$L_{3,28}$	9	24	0	60	0	**4.46**	14.70	14.74	**14.70**	3.30
$L_{13,053}$	30	13	1	20	0	**4.34**	426.13	274.33	**274.33**	63.21
$L_{13,058}$	30	13	1	24	0	**4.29**	402.95	416.64	**402.95**	93.91

$\mathcal{C}$	d	τ	s	e	a	**Isotop2**	CA0	CA1	**CA min**	$\frac{\text{CA}}{\text{Isotop2}}$
$L_{13,063}$	30	13	1	24	0	**4.24**	151.86	172.45	**151.86**	35.81
$L_{6,028}$	9	21	0	46	0	**4.24**	16.49	16.69	**16.49**	3.89
$M_{4,3,2}$	16	128	16	8	0	**4.23**	25.31	29.41	**25.31**	5.98
$L_{6,025}$	9	11	0	46	0	**4.21**	32.57	32.61	**32.57**	7.74
$L_{16,004}$	33	14	4	0	0	**4.20**	484.50	821.32	**484.50**	115.36
$M_{5,2,3}$	15	95	20	12	1	**4.20**	16.31	15.91	**15.91**	3.79
$L_{6,027}$	9	17	0	46	0	**4.19**	10.81	10.77	**10.77**	2.57
$L_{6,026}$	9	14	0	46	0	**4.18**	15.87	15.86	**15.86**	3.79
$L_{3,27}$	9	14	0	59	0	**4.16**	13.31	13.26	**13.26**	3.19
$L_{11,064}$	24	22	10	20	0	**4.00**	41.39	41.69	**41.39**	10.35
$L_{10,054}$	22	39	10	20	0	**3.96**	40.67	40.63	**40.63**	10.26
$L_{17,048}$	25	43	6	0	0	**3.96**	1161.36	1226.82	**1161.36**	293.27
$L_{17,061}$	33	16	2	0	0	**3.96**	2.06	5.71	**2.06**	0.52
$L_{10,054}$	22	39	10	20	0	**3.95**	40.65	40.61	**40.61**	10.28
$M_{4,3,4}$	16	127	12	8	1	**3.93**	26.03	25.93	**25.93**	6.60
$L_{2,63}$	48	7	1	0	0	**3.77**	6.02	43.37	**6.02**	1.60
$L_{3,26}$	9	7	0	58	0	**3.73**	40.18	40.16	**40.16**	10.77
$M_{4,3,3}$	16	130	6	8	0	**3.70**	25.21	27.22	**25.21**	6.81
$L_{17,045}$	25	70	2	0	0	**3.67**	1.77	3.74	**1.77**	0.48
$M_{7,1,5}$	14	64	4	8	2	**3.66**	11.09	8.65	**8.65**	2.36
$M_{5,2,4}$	15	96	10	18	0	**3.65**	15.72	15.40	**15.40**	4.22
$L_{13,052}$	28	12	1	20	0	**3.63**	143.42	154.63	**143.42**	39.51
$L_{9,011}$	24	14	12	20	0	**3.62**	297.68	297.92	**297.68**	82.21
$L_{11,054}$	22	22	10	20	0	**3.52**	34.00	34.62	**34.00**	9.66
$L_{13,030}$	24	10	1	12	0	**3.50**	56.56	103.52	**56.56**	16.16
$L_{10,045}$	20	39	10	20	0	**3.47**	32.43	32.41	**32.41**	9.34
$L_{2,81}$	40	3	1	0	0	**3.42**	0.93	8.30	**0.93**	0.27
$L_{10,045}$	20	39	10	20	0	**3.42**	32.31	32.18	**32.18**	9.41
$L_{13,048}$	30	13	1	20	0	**3.39**	650.08	77.45	**77.45**	22.84
$L_{1,7}$	10	8	36	1	0	**3.37**	15.30	19.43	**15.30**	4.54
$L_{13,057}$	28	12	1	24	0	**3.37**	158.91	154.16	**154.16**	45.74
$L_{11,059}$	24	20	5	10	0	**3.34**	20.90	20.46	**20.46**	6.13
$M_{5,2,2}$	15	98	4	6	0	**3.26**	10.93	12.19	**10.93**	3.35
$M_{7,1,3}$	14	64	4	8	1	**3.25**	12.48	11.48	**11.48**	3.53
$L_{16,008}$	33	7	4	0	0	**3.19**	1336.22	804.30	**804.30**	252.13
$L_{17,047}$	25	30	6	0	0	**3.12**	1160.49	1164.10	**1160.49**	371.95
$M_{5,2,1}$	15	95	23	6	0	**3.12**	17.81	18.02	**17.81**	5.71
$L_{13,062}$	28	12	1	24	0	**3.10**	102.22	91.52	**91.52**	29.51
$L_{1,6}$	9	7	28	1	0	**3.04**	9.10	9.12	**9.10**	2.99
$L_{17,044}$	25	56	2	0	0	**3.04**	1.43	3.03	**1.43**	0.47
$L_{11,045}$	20	22	10	20	0	**3.01**	27.48	28.07	**27.48**	9.13
$L_{11,062}$	24	17	8	16	0	**2.99**	21.36	21.39	**21.36**	7.14

$\mathcal{C}$	d	τ	s	e	a	**Isotop2**	CA0	CA1	**CA min**	$\frac{\text{CA}}{\text{Isotop2}}$
$M_{7,1,2}$	14	64	2	4	0	**2.98**	8.11	11.56	**8.11**	2.72
$L_{5,213}$	36	8	0	0	0	**2.97**	2.34	7.11	**2.34**	0.79
$L_{2,58}$	40	7	1	0	0	**2.96**	6.12	30.72	**6.12**	2.07
$L_{13,047}$	28	12	1	20	0	**2.95**	260.30	338.01	**260.30**	88.21
$L_{3,24}$	8	34	0	47	0	**2.92**	6.47	6.49	**6.47**	2.22
$L_{2,54}$	40	8	1	0	0	**2.89**	9.22	46.30	**9.22**	3.19
$L_{13,051}$	26	11	1	20	0	**2.87**	148.67	110.44	**110.44**	38.48
$L_{10,044}$	20	34	9	18	0	**2.84**	20.41	20.34	**20.34**	7.16
$L_{11,049}$	22	20	5	10	0	**2.81**	16.62	16.87	**16.62**	5.91
$M_{7,1,1}$	14	65	6	8	0	**2.80**	15.30	14.86	**14.86**	5.31
$L_{13,056}$	26	11	1	24	0	**2.74**	22.07	128.00	**22.07**	8.05
$L_{5,204}$	30	18	0	0	0	**2.72**	0.78	2.74	**0.78**	0.29
$L_{3,25}$	8	51	0	48	0	**2.69**	7.26	7.29	**7.26**	2.70
$L_{11,052}$	22	17	8	16	0	**2.67**	17.52	17.62	**17.52**	6.56
$L_{3,23}$	8	21	0	46	0	**2.67**	5.56	5.61	**5.56**	2.08
$L_{17,043}$	25	43	2	0	0	**2.63**	1.24	2.68	**1.24**	0.47
$L_{13,061}$	26	11	1	24	0	**2.61**	35.42	50.49	**35.42**	13.57
$L_{10,044}$	20	34	9	18	0	**2.60**	20.02	20.44	**20.02**	7.70
$L_{6,022}$	8	13	0	37	0	**2.59**	5.00	4.93	**4.93**	1.90
$M_{4,3,1}$	16	129	6	0	0	**2.55**	22.40	20.50	**20.50**	8.04
$L_{3,22}$	8	11	0	45	0	**2.53**	5.03	5.02	**5.02**	1.98
$L_{6,021}$	8	10	0	37	0	**2.52**	4.84	4.87	**4.84**	1.92
$L_{13,041}$	26	11	1	16	0	**2.52**	125.30	92.64	**92.64**	36.76
$L_{11,040}$	20	20	5	10	0	**2.51**	13.75	14.04	**13.75**	5.48
$L_{6,064}$	10	22	0	13	0	**2.51**	5.55	5.50	**5.50**	2.19
$L_{10,062}$	24	29	8	16	0	**2.50**	24.53	24.96	**24.53**	9.81
$L_{6,063}$	10	18	0	13	0	**2.49**	6.88	6.95	**6.88**	2.76
$L_{6,062}$	10	15	0	13	0	**2.47**	5.38	5.34	**5.34**	2.16
$L_{6,023}$	8	17	0	37	0	**2.47**	4.95	4.96	**4.95**	2.00
$L_{2,67}$	42	5	1	0	0	**2.46**	2.13	15.41	**2.13**	0.87
$L_{13,046}$	26	11	1	20	0	**2.44**	144.56	149.34	**144.56**	59.25
$L_{6,061}$	10	12	0	13	0	**2.43**	14.97	14.92	**14.92**	6.14
$L_{11,057}$	24	21	3	6	0	**2.40**	17.25	17.59	**17.25**	7.19
$L_{17,046}$	25	16	6	0	0	**2.39**	Timeout	Timeout	**Timeout**	>753
$L_{6,024}$	8	20	0	37	0	**2.38**	4.95	5.00	**4.95**	2.08
$L_{13,055}$	24	10	1	24	0	**2.36**	25.17	29.46	**25.17**	10.67
$L_{9,010}$	22	14	10	12	0	**2.36**	88.61	101.35	**88.61**	37.55
$L_{10,062}$	24	29	8	16	0	**2.33**	24.31	24.63	**24.31**	10.43
$L_{9,009}$	20	13	10	20	0	**2.31**	138.71	131.60	**131.60**	56.97
$L_{10,035}$	18	29	8	16	0	**2.30**	3.11	3.20	**3.11**	1.35
$L_{3,21}$	8	7	0	44	0	**2.30**	4.18	4.26	**4.18**	1.82
$L_{11,056}$	24	17	2	4	0	**2.24**	11.96	12.34	**11.96**	5.34

$\mathcal{C}$	d	τ	s	e	a	**Isotop2**	CA0	CA1	**CA min**	$\frac{\text{CA}}{\text{Isotop2}}$
$L_{11,048}$	22	15	4	8	0	**2.23**	8.70	8.88	**8.70**	3.90
$L_{5,203}$	30	15	0	0	0	**2.23**	1.35	3.28	**1.35**	0.61
$L_{2,76}$	36	3	1	0	0	**2.16**	0.62	4.91	**0.62**	0.29
$L_{10,060}$	24	41	6	12	0	**2.15**	42.89	44.09	**42.89**	19.95
$L_{10,060}$	24	41	6	12	0	**2.15**	43.02	43.33	**43.02**	20.01
$L_{17,042}$	25	30	2	0	0	**2.14**	0.96	2.11	**0.96**	0.45
$L_{10,052}$	22	29	8	16	0	**2.13**	20.41	20.02	**20.02**	9.40
$L_{11,043}$	20	17	8	16	0	**2.12**	14.21	14.26	**14.21**	6.70
$L_{11,039}$	20	15	4	8	0	**2.06**	7.24	7.19	**7.19**	3.49
$L_{10,052}$	22	29	8	16	0	**2.06**	19.76	20.06	**19.76**	9.59
$L_{13,036}$	26	11	1	16	0	**2.01**	32.81	147.63	**32.81**	16.32
$M_{6,1,1}$	12	65	6	12	0	**1.96**	6.17	5.81	**5.81**	2.96
$L_{11,050}$	22	12	6	12	0	**1.94**	7.58	7.79	**7.58**	3.91
$L_{3,20}$	7	54	0	36	0	**1.92**	1.34	1.33	**1.33**	0.69
$L_{13,040}$	24	10	1	16	0	**1.92**	76.44	60.96	**60.96**	31.77
$L_{13,045}$	24	10	1	20	0	**1.90**	104.51	23.20	**23.20**	12.21
$L_{13,050}$	24	10	1	20	0	**1.89**	67.56	52.66	**52.66**	27.86
$L_{11,035}$	18	17	8	16	0	**1.88**	11.20	11.55	**11.20**	5.95
$L_{10,056}$	24	28	2	2	0	**1.88**	7.09	7.52	**7.09**	3.77
$M_{6,1,2}$	12	65	2	4	1	**1.84**	3.31	3.08	**3.08**	1.67
$M_{4,2,3}$	12	96	12	6	1	**1.83**	3.23	3.58	**3.23**	1.77
$L_{3,19}$	7	37	0	35	0	**1.82**	1.14	1.13	**1.13**	0.62
$L_{10,043}$	20	29	8	16	0	**1.81**	10.58	10.85	**10.58**	5.85
$L_{3,18}$	7	24	0	34	0	**1.81**	0.97	0.98	**0.97**	0.54
$L_{10,056}$	24	28	2	2	0	**1.80**	7.05	7.42	**7.05**	3.92
$L_{10,043}$	20	29	8	16	0	**1.79**	10.72	10.76	**10.72**	5.99
$L_{10,035}$	18	29	8	16	0	**1.77**	3.67	3.16	**3.16**	1.79
$L_{6,020}$	7	19	0	26	0	**1.75**	0.78	0.80	**0.78**	0.45
$L_{3,17}$	7	14	0	33	0	**1.75**	2.11	2.11	**2.11**	1.21
$L_{13,044}$	22	9	1	20	0	**1.71**	35.88	38.70	**35.88**	20.97
$L_{5,202}$	30	11	0	0	0	**1.71**	1.14	2.96	**1.14**	0.67
$L_{10,059}$	24	31	5	10	0	**1.70**	24.47	26.00	**24.47**	14.39
$L_{11,031}$	18	15	4	8	0	**1.70**	5.67	5.76	**5.67**	3.34
$L_{13,035}$	24	10	1	16	0	**1.67**	77.96	80.22	**77.96**	46.71
$L_{16,013}$	25	3	4	0	0	**1.66**	822.08	849.64	**822.08**	495.23
$L_{11,028}$	16	17	8	16	0	**1.64**	17.67	18.10	**17.67**	10.77
F_{24}	16	22	9	8	0	**1.63**	10.01	7.81	**7.81**	4.79
$L_{1,5}$	8	7	21	1	0	**1.63**	1.48	1.19	**1.19**	0.73
$L_{10,028}$	16	29	8	16	0	**1.63**	9.50	9.54	**9.50**	5.83
$L_{11,046}$	22	14	2	4	0	**1.62**	6.85	7.08	**6.85**	4.23
$L_{3,16}$	7	7	0	32	0	**1.62**	1.82	1.83	**1.82**	1.12
$L_{10,058}$	24	36	4	6	0	**1.60**	21.54	22.27	**21.54**	13.46

$\mathcal{C}$	d	τ	s	e	a	**Isotop2**	CA0	CA1	**CA min**	$\frac{\text{CA}}{\text{Isotop2}}$
$L_{13,039}$	22	9	1	16	0	**1.60**	39.62	32.83	**32.83**	20.52
$L_{10,028}$	16	29	8	16	0	**1.59**	9.40	9.48	**9.40**	5.91
$L_{11,041}$	20	12	6	12	0	**1.58**	6.07	6.34	**6.07**	3.84
$M_{4,2,1}$	12	99	6	9	1	**1.58**	2.65	2.95	**2.65**	1.68
$L_{13,049}$	22	9	1	20	0	**1.58**	16.08	19.83	**16.08**	10.18
$L_{17,030}$	17	70	6	0	0	**1.58**	22.09	21.64	**21.64**	13.70
$L_{6,017}$	7	9	0	26	0	**1.58**	1.96	1.95	**1.95**	1.23
$L_{13,029}$	22	9	1	12	0	**1.57**	37.08	53.71	**37.08**	23.62
$L_{10,058}$	24	36	4	6	0	**1.55**	21.13	21.35	**21.13**	13.63
$L_{6,018}$	7	12	0	26	0	**1.52**	1.96	2.00	**1.96**	1.29
$L_{6,019}$	7	16	0	26	0	**1.52**	2.09	2.22	**2.09**	1.38
$L_{17,041}$	25	16	2	0	0	**1.52**	0.72	1.75	**0.72**	0.47
$L_{10,059}$	24	31	5	10	0	**1.51**	23.64	23.87	**23.64**	15.65
$L_{17,020}$	17	140	6	0	0	**1.48**	27.11	29.39	**27.11**	18.33
$L_{6,060}$	9	21	0	12	0	**1.47**	2.93	2.95	**2.93**	1.99
F_1	8	6	21	7	0	**1.47**	0.86	0.85	**0.85**	0.58
$L_{6,059}$	9	17	0	12	0	**1.46**	2.88	2.91	**2.88**	1.97
$L_{11,037}$	20	14	2	4	0	**1.45**	5.52	5.69	**5.52**	3.80
$L_{17,029}$	17	56	6	0	0	**1.45**	19.65	15.48	**15.48**	10.68
$L_{6,057}$	9	11	0	12	0	**1.45**	7.70	7.69	**7.69**	5.30
$L_{11,024}$	16	15	4	8	0	**1.44**	13.01	13.20	**13.01**	9.03
$L_{6,058}$	9	14	0	12	0	**1.44**	2.87	2.94	**2.87**	1.99
$M_{4,2,5}$	12	96	10	6	0	**1.43**	3.17	3.52	**3.17**	2.22
$L_{10,049}$	22	31	5	10	0	**1.42**	19.59	20.08	**19.59**	13.79
$L_{1,4}$	7	6	15	1	0	**1.41**	0.74	0.76	**0.74**	0.52
$L_{13,034}$	22	9	1	16	0	**1.41**	67.66	36.48	**36.48**	25.87
$L_{13,043}$	20	8	1	20	0	**1.39**	5.60	16.11	**5.60**	4.03
$L_{10,049}$	22	31	5	10	0	**1.34**	19.40	19.67	**19.40**	14.48
$M_{5,1,3}$	10	64	6	12	0	**1.32**	2.92	3.05	**2.92**	2.21
$L_{5,201}$	30	8	0	0	0	**1.31**	1.14	2.90	**1.14**	0.87
$L_{10,057}$	24	32	3	2	0	**1.27**	7.30	7.72	**7.30**	5.75
$L_{10,057}$	24	32	3	2	0	**1.27**	7.40	7.81	**7.40**	5.83
$L_{2,71}$	32	3	1	0	0	**1.27**	0.39	2.70	**0.39**	0.31
$M_{4,2,4}$	12	96	8	6	0	**1.27**	2.41	2.54	**2.41**	1.90
$L_{11,033}$	18	12	6	12	0	**1.25**	4.84	5.15	**4.84**	3.87
$L_{10,040}$	20	31	5	10	0	**1.24**	10.78	11.08	**10.78**	8.69
$L_{17,019}$	17	113	6	0	0	**1.24**	20.26	17.59	**17.59**	14.19
$M_{6,1,5}$	12	65	3	6	1	**1.23**	3.03	4.67	**3.03**	2.46
$L_{17,028}$	17	43	6	0	0	**1.23**	17.44	13.36	**13.36**	10.86
$L_{13,024}$	22	9	1	12	0	**1.22**	68.10	38.29	**38.29**	31.39
$L_{3,15}$	6	52	0	25	0	**1.22**	0.64	0.65	**0.64**	0.52
$L_{10,050}$	22	19	6	12	0	**1.21**	8.10	8.21	**8.10**	6.69

$\mathcal{C}$	d	τ	s	e	a	**Isotop2**	CA0	CA1	**CA min**	$\frac{\text{CA}}{\text{Isotop2}}$
$L_{13,028}$	20	8	1	12	0	**1.20**	8.45	31.76	**8.45**	7.04
$L_{10,040}$	20	31	5	10	0	**1.19**	10.76	10.97	**10.76**	9.04
$L_{2,62}$	36	5	1	0	0	**1.19**	1.24	6.97	**1.24**	1.04
$M_{6,1,4}$	12	66	3	6	1	**1.18**	2.46	2.92	**2.46**	2.08
$L_{16,003}$	25	10	4	0	0	**1.16**	30.77	95.26	**30.77**	26.50
$L_{17,018}$	17	86	6	0	0	**1.16**	12.03	13.52	**12.03**	10.36
$L_{2,53}$	32	7	1	0	0	**1.15**	2.45	10.90	**2.45**	2.13
$L_{13,033}$	20	8	1	16	0	**1.14**	22.12	36.60	**22.12**	19.40
$L_{5,192}$	24	18	0	0	0	**1.14**	0.42	1.00	**0.42**	0.37
$L_{10,050}$	22	19	6	12	0	**1.13**	8.44	8.20	**8.20**	7.25
$L_{13,038}$	20	8	1	16	0	**1.13**	21.64	23.29	**21.64**	19.13
$L_{17,027}$	17	30	6	0	0	**1.11**	4.19	15.11	**4.19**	3.77
$L_{3,14}$	6	36	0	25	0	**1.09**	0.54	0.52	**0.52**	0.48
$L_{11,015}$	12	12	6	12	0	**1.07**	5.92	6.00	**5.92**	5.53
$L_{17,015}$	17	140	2	0	0	**1.06**	2.68	3.64	**2.68**	2.53
$L_{11,026}$	16	12	6	12	0	**1.06**	11.18	11.31	**11.18**	10.55
$M_{4,2,2}$	12	94	5	3	0	**1.05**	2.33	2.62	**2.33**	2.22
$L_{10,041}$	20	19	6	12	0	**1.05**	4.14	4.45	**4.14**	3.94
$L_{13,018}$	20	8	1	8	0	**1.03**	32.35	7.44	**7.44**	7.22
$L_{3,12}$	6	12	0	23	0	**1.03**	0.81	0.80	**0.80**	0.78
$L_{9,007}$	16	7	8	12	0	**1.02**	4.38	5.51	**4.38**	4.29
$L_{10,048}$	22	23	4	6	0	**1.01**	6.48	6.57	**6.48**	6.41
$L_{17,017}$	17	60	6	0	0	**1.01**	6.85	9.23	**6.85**	6.78
$L_{13,037}$	18	7	1	16	0	**1.00**	2.51	5.24	**2.51**	2.51
$M_{5,1,4}$	10	65	4	8	0	**1.00**	1.98	2.25	**1.98**	1.98
$L_{10,046}$	22	23	2	2	0	**1.00**	3.92	4.12	**3.92**	3.92
$L_{10,046}$	22	23	2	2	0	**1.00**	3.92	4.12	**3.92**	3.92
$L_{5,191}$	24	15	0	0	0	**1.00**	0.66	1.27	**0.66**	0.66

Curves	RS3_Seq	**Iso2_Seq**	Topology	RS3_MT3	**Iso2_MT3**	**Fast-Ana**	$\frac{\text{Iso2}}{\text{Fast-Ana}}$	$\frac{\text{Iso2\_MT3}}{\text{Fast-Ana}}$
challenge_12b	19.38	**58.82**	39.44	10.31	**49.75**	**44.3**	1.32	1.12
degree_7_surf	12.73	**17.734**	5.004	5.72	**10.724**	**7.39**	2.40	1.45
FTT_5_4_4	14.82	**57.22**	42.4	7.45	**49.85**	**32.23**	1.77	1.54
L6_circle	3.82	**6.32**	2.5	1.81	**4.31**	**2.1**	3	2.05
mignotte	233	**345**	112	165.3	**277.3**	**136.54**	2.53	2.03
spider	96.05	**167.77**	71.72	42.57	**114.29**	**26.26**	6.35	4.35
swinnerton	1.77	**6.8**	5.03	0.97	**6**	**7.12**	0.95	0.84

Table 7.12: Benchmark results of Isotop2 and *FastAnalysis* run on the set of curves from [Berberich 2011a]. The column labelled Topology shows the running time of the topology part in Isotop2 that is, after isolating boxes for the critical points have been computed. Running times are in seconds.

The following figures contain graphs (generated by running Isotop2 on Maple) corresponding to the polynomials in Table 7.12.

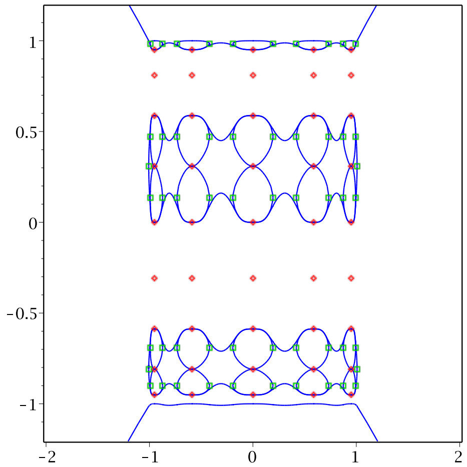

Figure 7.1: challenge_12b

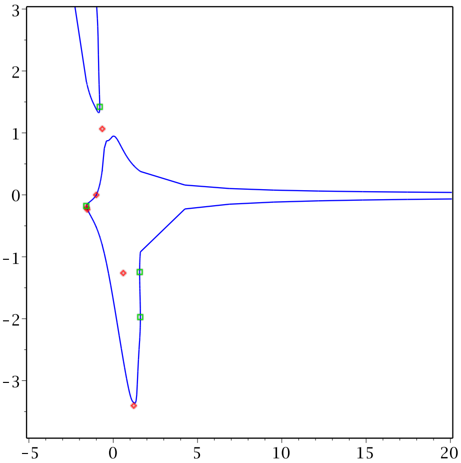

Figure 7.2: degree_7_surf

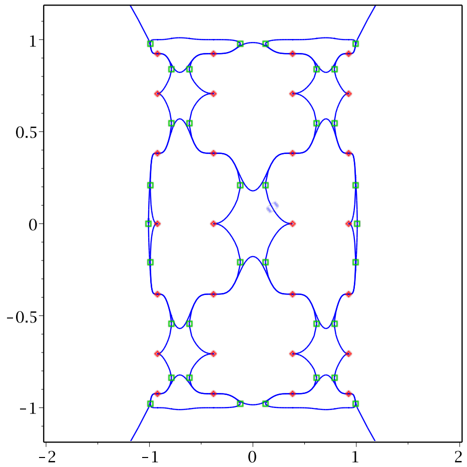

Figure 7.3: FTT_5_4_4

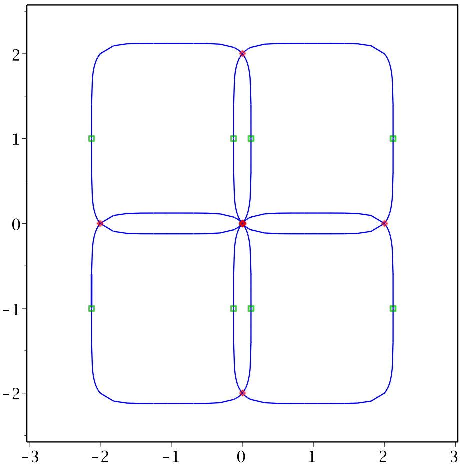

Figure 7.4: L6_circles

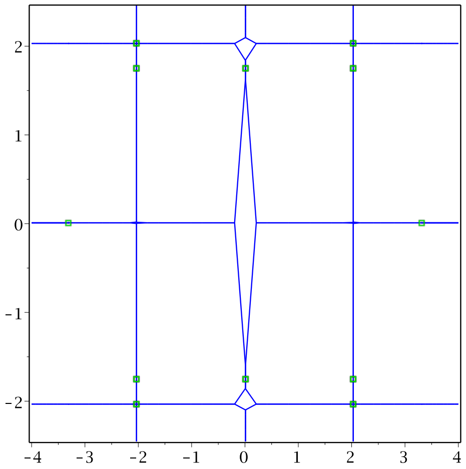

Figure 7.5: mignotte

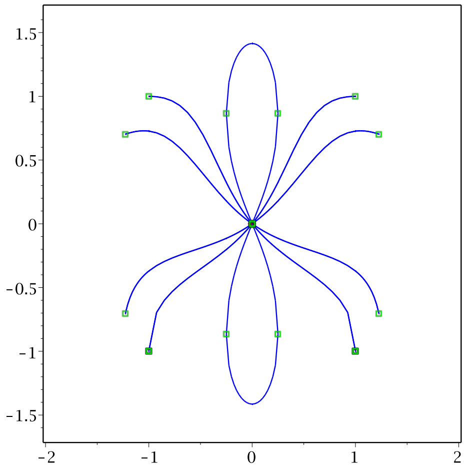

Figure 7.6: spider

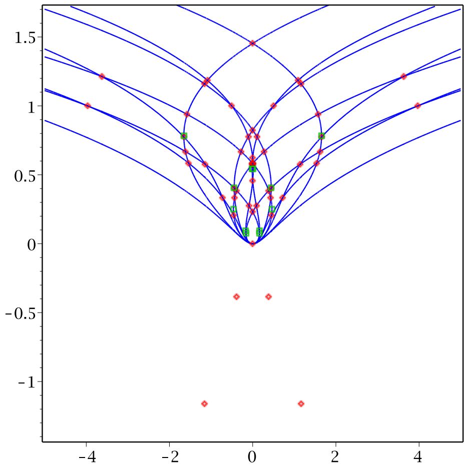

Figure 7.7: swinnerton

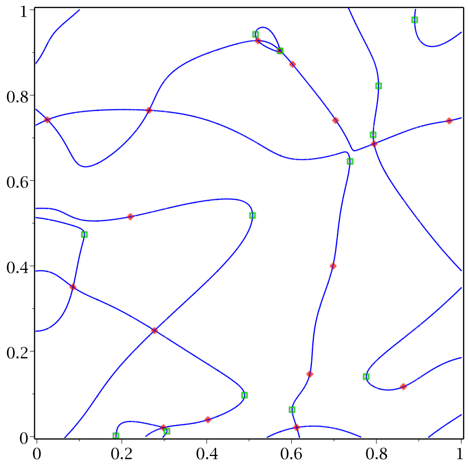

Figure 7.8: ridge

Chapter 8
Conclusion

We addressed in this thesis the problem of solving bivariate systems with integer coefficients and its application for computing the topology of real plane curves. While focusing on the computation of a Rational Univariate Representation of the solutions of such systems, we tackled this problem from two different points of view: the theoretical bit complexity and the practical efficiency.

After investigating in-depth the different steps in the computation of the Rational Univariate Representation of bivariate systems, we presented two algorithms of worst-case bit-complexities in $\widetilde{O}_B(d^8+d^7\tau)$ and in $\widetilde{O}_B(d^7+d^6\tau)$ for respectively computing a separating linear form and, given such a form, a Rational Univariate Representation of the solutions of a bivariate system. While the first algorithm improves the best known complexity bound by a factor d^2, the second one provide simple formulas for the RUR polynomials that allow us to derive a new bound on the size of the coefficients in the Rational Univariate Representation of a bivariate system.

This bound turns out to be of great interest in the design of an efficient Chinese Remainder Theorem-based practical algorithm for computing the RURs of a bivariate system. The idea of this algorithm is to compute in a multi-modular fashion, a decomposition of the bivariate system into a set of RURs by first computing a triangular decomposition [Gonzalez-Vega 1996] and then computing Rational Univariate Representations for the resulting triangular systems. As usual, an issue that is inherent to the use of multi-modular computation is the correctness of the result. We overcome this issue by proposing two methods for checking that the computed RURs encode the solutions of the input system.

At first glance, computing a RUR after a triangular decomposition may seem superfluous and time consuming. However, our experiments show that first, computing a RUR does not induce a large overhead compared to only computing a triangular decomposition, and second, it avoids the difficult isolation of the solutions in fibers in the non-radical or non-generic cases. This practical behavior is in accordance with the complexity analysis which shows that the complexity of the computation of the RURs is the same as the complexity of the computation of the triangular decomposition. More generally, we prove an expected complexity bound in $\widetilde{O}_B(d^6 + d^5\tau)$ for this practical algorithm.

From the complexity point of view, although the previous bounds are quite remarkable (the expected complexity of our practical algorithm matches that for only the isolation of the roots of the squarefree part of the resultant), an ongoing work seems to suggest that for the specific case of systems defined by a polynomial and one of its partial derivatives, as for instance, the system of the critical points of a curve, some of the previous complexity bounds could be improved by a factor d. This suggests, in other words, that computing the number of critical points of a curve can be achieved in a better complexity than that of computing the number of distinct solutions of a general bivariate system (namely for the cost of a triangular decomposition).

More precisely, if the system at hand is given by a polynomial f and its derivative with respect to y, f_y, the idea roughly is to perform the triangular decompositions of the two systems $\{f, f_y\}$ and $\{f, f_y^2\}$ and to compute the difference between the number of solutions counted with multiplicities in the resulting triangular systems. As the solutions of $\{f, f_y^2\}$ have their multiplicities in fiber larger by one than the multiplicities in fiber of the solutions of $\{f, f_y\}$, computing the difference yields exactly the number of distinct solutions. In addition, this computation can be done without computing the whole triangular systems but only the leading terms of the latter, which amounts to only compute the sequence of the principal subresultant coefficients. This results in an algorithm with arithmetic complexity in $\widetilde{O}(d^3)$ for computing the number of distinct solutions of the system $\{f, f_y\}$. Replacing in Algorithm 3 of Chapter 3, the existing algorithm for computing the number of distinct solutions, by the one above, reduces the worst-case bit complexity for computing a separating linear form for the system $\{f, f_y\}$ to $\widetilde{O}_B(d^7 + d^6\tau)$. In addition, one can perform the previous computation in a multi-modular Las-Vegas fashion which reduces the complexity of computing a separating linear form to $\widetilde{O}(d^5 + d^4\tau)$ in expected setting. Also, following the approach of [Gonzalez-Vega 1996], it is most likely that, within the same expected bit complexity, one can compute a decomposition of the system into a set of subresultant-based rational parametrizations. Beside, we are convinced that with some work, the previous results can be extended to the case of general bivariate systems.

From the practical point of view, as mentioned in Chapter 5, in the current implementation, only the verification based on the subtitution of the rational fractions of the RUR in the input polynomials is available. Although this strategy has shown quite good performance in practice, it will be interesting to investigate other methods for checking the correctness of the output of our bivariate solver. One of these methods, already mentionned in Chapter 5, consists in using the approach of Daouda et al. [Niang Diatta 2008] that checks the result directly on the resulting triangular system. Another promising method is to use the numerical inclusion predicate presented in [Berberich 2011b], which ensures the existence of a unique solution inside a given box. This predicate have been shown to be very efficient in practice. In addition, unlike in [Berberich 2011b] where the predicate is used for a set of d^4 candidate solutions in the worst case, in our case, after the RUR-candidates computation, we end up with at most d^2 candidate solutions, which reduces substantially the overall cost of the check.

Another important point to further improve our implementation concerns the algorithms used for basic operations. Indeed, in our theoretical description, we use algorithms that are known to have the best complexity bounds. This is the case for example for the computation of the gcd of univariate polynomials or the computation of one subresultant polynomial. However, in practice, due to the constant factors hidden in the complexity bounds, such algorithms become really efficient (compared to the naive algorithms) only from a certain degree threshold. As the degree of the polynomials computed in our algorithm, considerably varies during the algorithm, we thus aim at designing algorithms for basic operations, which depending on the degree of the input polynomials choose the most efficient strategy.

Finally, we can identify the following directions for future work. First, our recent improvements for solving bivariate systems yield that the corresponding step is most often non-dominant in the whole topology computation, as shown in our experiments (Table 7.12). It is thus interesting to work again on the other parts of the topology computation algorithm in order to improve their efficiency. In addition, the complexity analysis of the topology algorithm performed in [Cheng 2010] yields a bound of $\widetilde{O}_B(d^{24}\tau^2)$ that is most likely very pessimistic. A more careful analysis, at the light of the results obtained in this thesis, will undoubtedly reduces the latter to more or less that of the bivariate system solving as hinted by the work of Mehlhorn et al. [Mehlhorn 2013].

Given an efficient algorithm for computing the topology of a curve, a natural problem is consider arrangements of curves. For the time being, our algorithm Isotop2 is only able to handle single curves. In other words, if the input curve is given as a product of several curves, our algorithm considers it as a unique curve. However, it is quite clear that computing the topology of every curve and combining them afterward is much more efficient that computing the topology of the product. This problem of computing the topology of the arrangement of curves knowing their topologies was studied in detail by Kerber in his thesis [Kerber 2009a] using a CAD like approach. We want, in our case, to tackle it using a RUR-based approach. Though it seems at a first glance, that having the critical points under RUR representation should ease many operations in the arrangement algorithm, a more in-depth study is however needed to evaluate precisely the relevance of a RUR-based approach for computing the arrangement of algebraic curves.

Finally, a quite natural and relevant direction of research is the study of algorithms handling algebraic objects in three dimensions such as curves and surfaces. More precisely, we aim at computing a graph isotopic to a 3D curve or a piecewise linear mesh isotopic to a surface. Computing the topology of a curve, which could be the singular locus of a surface, can be done by projecting it onto a plane, computing the topology of the projection, and then lifting back to recover the desired topology in 3D. It can also be done by considering the curve directly in 3D, which amounts to computing with polynomial systems in three variables. The first approach has already been a focus of interest (see [Berberich 2010] and references therein). The drawback of such an approach, however, is that the projection step often creates spurious singular points. Investigating the second approach from the complexity and the practical efficiency points of view is thus of particular interest.

Bibliography

[Alberti 2008] L. Alberti, B. Mourrain and J. Wintz. *Topology and arrangement computation of semi-algebraic planar curves.* Comput. Aided Geom. Des., vol. 25, no. 8, pages 631–651, 2008. (Cited on page 18.)

[Alonso 1996] M.-E Alonso, E. Becker, M.-F. Roy and T. Wörmann. *Multiplicities and Idempotents for Zerodimensional Systems.* In Algorithms in Algebraic Geometry and Applications, volume 143 of *Progress in Mathematics*, pages 1–20. Birkhäuser, 1996. (Cited on pages 16, 62 and 79.)

[Aubry 1999a] P. Aubry, D. Lazard and M. Moreno Maza. *On the theories of triangular sets.* J. Symb. Comput., vol. 28, pages 105–124, July 1999. (Cited on page 15.)

[Aubry 1999b] P. Aubry and M. Moreno Maza. *Triangular sets for solving polynomial systems: a comparative implementation of four methods.* J. Symb. Comput., vol. 28, pages 125–154, July 1999. (Cited on page 18.)

[Basu 2006] S. Basu, R. Pollack and M.-F. Roy. Algorithms in real algebraic geometry, volume 10 of *Algorithms and Computation in Mathematics*. Springer-Verlag, 2nd édition, 2006. (Cited on pages 19, 24, 25, 26, 27, 32, 33, 34, 45, 49, 90, 94, 114 and 116.)

[Berberich 2010] Eric Berberich, Michael Kerber and Michael Sagraloff. *An efficient algorithm for the stratification and triangulation of an algebraic surface.* Comput. Geom. Theory Appl., vol. 43, no. 3, pages 257–278, 2010. (Cited on page 154.)

[Berberich 2011a] E. Berberich, P. Emeliyanenko, A. Kobel and M. Sagraloff. *Arrangement Computation for Planar Algebraic Curves.* In Symbolic Numeric Computation - SNC, 2011. Arxiv preprint http://arxiv.org/abs/1103.4697. (Cited on pages 17, 131, 132, 137 and 148.)

[Berberich 2011b] E. Berberich, P. Emeliyanenko and M. Sagraloff. *An Elimination Method for Solving Bivariate Polynomial Systems: Eliminating the Usual Drawbacks.* In Alenex, 2011. (Cited on pages 14, 131, 132 and 153.)

[Berberich 2013] Eric Berberich, Michael Hemmer, Michael Kerber, Sylvain Lazard, Luis Peñaranda and Monique Teillaud. *Algebraic Kernel.* In CGAL User and Reference Manual. CGAL Editorial Board, 4.3 édition, 2013. (Cited on page 128.)

[Boulier 2009] F. Boulier, C. Chen, F. Lemaire and M. Moreno Maza. *Real Root Isolation of Regular Chains.* In Proceedings of the 2009 Asian Symposium on Computer Mathematics (ASCM 2009), Math for Industry, pages 1–15, 2009. (Cited on page 15.)

[Bouzidi 2011] Y. Bouzidi, S. Lazard, M. Pouget and F. Rouillier. *New bivariate system solver and topology of algebraic curves.* In 27th European Workshop on Computational Geometry - EuroCG, 2011. (Cited on page 76.)

[Bouzidi 2013a] Y. Bouzidi, S. Lazard, M. Pouget and F. Rouillier. *Rational Univariate Representations of Bivariate Systems and Applications.* In Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation - ISSAC'13, 2013. (Cited on page 61.)

[Bouzidi 2013b] Y. Bouzidi, S. Lazard, M. Pouget and F. Rouillier. *Separating linear forms for bivariate systems.* In Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation - ISSAC'13, 2013. (Cited on pages 41, 106 and 109.)

[Brown 1971] W. S. Brown. *On Euclid's Algorithm and the Computation of Polynomial Greatest Common Divisors.* Journal of the ACM, vol. 18, pages 478–504, 1971. (Cited on pages 35 and 39.)

[Brown 2002] C. W. Brown. *Contructing Cylindrical Algebraic Decomposition of the Plane Quickly*, 2002. Manuscript, `http://www.cs.usna.edu/~wcbrown/`. (Cited on pages 131 and 140.)

[Burr 2008] M. Burr, S.W.Choi, B. Galehouse and C. Yap. *Complete Subdivision Algorithms, II: Isotopic Meshing of Singular Algebraic Curves.* In Proceeding of the 33th International Symposium on Symbolic and Algebraic Computation - ISSAC, 2008. (Cited on pages 17 and 18.)

[Busé 2005] L. Busé, H. Khalil and B. Mourrain. *Resultant-based methods for plane curves intersection problems.* In Computer Algebra in Scientific Computing (CASC), volume 3718 of *Lecture Notes in Computer Science*, pages 75–92, Kalamata, Greece, September 2005. Springer Berlin / Heidelberg. (Cited on page 66.)

[Canny 1987] J. Canny. *A new algebraic method for robot motion planning and real geometry.* In Proceedings of the 28th Annual Symposium on Foundations of Computer Science, SFCS'87, pages 39–48, Washington, DC, USA, 1987. IEEE Computer Society. (Cited on page 62.)

[Canny 1988] J. Canny. *Some algebraic and geometric computations in PSPACE.* In Proceedings of the twentieth annual ACM symposium on Theory of computing, STOC '88, pages 460–467, New York, NY, USA, 1988. ACM. (Cited on page 16.)

[Cazals 2008] F. Cazals, J.-C. Faugère, M. Pouget and F. Rouillier. *Ridges and Umbilics of Polynomial Parametric Surfaces.* In B. Juttler and R. Piene, editeurs, Geometric Modeling and Algebraic Geometry, chapitre 3, pages 141–159. Springer, 2008. (Cited on pages 131 and 138.)

[cga] CGAL., *Computational Geometry Algorithms Library.* http://www.cgal.org. (Cited on page 124.)

[Chen 2011] C. Chen and M. Moreno Maza. *Algorithms for computing triangular decompositions of polynomial systems.* In Proceeding of the 36th International Symposium on Symbolic and Algebraic Computation, ISSAC '11, pages 83–90, New York, NY, USA, 2011. ACM. (Cited on page 15.)

[Cheng 2007] J.-S. Cheng, X.-S. Gao and C. K. Yap. *Complete numerical isolation of real zeros in zero-dimensional triangular systems.* In Proceeding of the International Symposium on Symbolic and Algebraic Computation- ISSAC, pages 92–99, 2007. (Cited on page 15.)

[Cheng 2009] J.-S. Cheng, X.-S. Gao and J. Li. *Root isolation for bivariate polynomial systems with local generic position method.* In ISSAC, pages 103–110, 2009. (Cited on pages 13 and 132.)

[Cheng 2010] J.-S. Cheng, S. Lazard, L. Peñaranda, M. Pouget, F. Rouillier and E. Tsigaridas. *On the Topology of Real Algebraic Plane Curves.* Mathematics in Computer Science, vol. 4, pages 113–137, 2010. (Cited on pages 11, 17, 18, 105, 119, 131, 137, 140 and 153.)

[Collins 1971] G. E. Collins. *The calculation of multivariate polynomial resultants.* In Proceedings of the second ACM symposium on Symbolic and algebraic manipulation, SYMSAC '71, pages 212–222, New York, NY, USA, 1971. ACM. (Cited on page 35.)

[Cox 1997] D. Cox, J. Little and D. O'Shea. Ideals, varieties, and algorithms. Undergraduate Texts in Mathematics. Springer-Verlag, New York, 2nd édition, 1997. (Cited on page 30.)

[Cox 2005] D. Cox, J. Little and D. O'Shea. Using algebraic geometry. Numeéro 185 de Graduate Texts in Mathematics. Springer, New York, 2nd édition, 2005. (Cited on pages 78 and 83.)

[Cox 2007] D. Cox, J. Little and D. O'Shea. Ideals, varieties, and algorithms. Undergraduate Texts in Mathematics. Springer-Verlag, New York, 3rd édition, 2007. (Cited on page 83.)

[Dahan 2004] X. Dahan and É. Schost. *Sharp estimates for triangular sets.* In Proceedings of the 2004 International Symposium on Symbolic and algebraic computation, ISSAC'04, pages 103–110, New York, NY, USA, 2004. ACM. (Cited on page 15.)

[Diatta 2009] D. Niang Diatta. *Calcul effectif de la topologie de courbes et surfaces algebriques reelles.* Ph.d. thesis, Universite de Limoge, France, 2009. (Cited on pages 78 and 102.)

[Diochnos 2009] 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., vol. 44, no. 7, pages 818–835, 2009. (Cited on pages 9, 10, 11, 13, 14, 15, 17, 42, 45, 51, 59, 76, 83, 112 and 117.)

[Durvye 2008] Clémence Durvye and Grégoire Lecerf. *A concise proof of the Kronecker polynomial system solver from scratch.* Expositiones Mathematicae, vol. 26, no. 2, pages 101–139, 2008. (Cited on page 16.)

[Eigenwillig 2007] A. Eigenwillig, M. Kerber and N. Wolpert. *Fast and Exact Geometric Analysis of Real Algebraic Plane Curves.* In Proceedings of the International Symposium on Symbolic and Algebraic Computation - ISSAC, pages 151–158, 2007. (Cited on pages 18, 128 and 137.)

[Eigenwilling 2008] A. Eigenwilling and M. Kerber. *Exact and Efficient 2D-Arrangements of Arbitrary Algebraic Curves.* In Proc. 19th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA08), pages 122–131, San Francisco, USA, January 2008. ACM-SIAM, ACM/SIAM. (Cited on page 128.)

[El Kahoui 2003] M. El Kahoui. *An elementary approach to subresultants theory.* J. Symb. Comput., vol. 35, no. 3, pages 281–292, 2003. (Cited on pages 31, 32, 34 and 114.)

[Emeliyanenko 2010] Pavel Emeliyanenko. *Modular resultant algorithm for graphics processors.* In Proceedings of the 10th international conference on Algorithms and Architectures for Parallel Processing - Volume Part I, ICA3PP'10, pages 427–440, Berlin, Heidelberg, 2010. Springer-Verlag. (Cited on page 14.)

[Emeliyanenko 2011] P. Emeliyanenko and M. Sagraloff. *On the Complexity of Solving a Bivariate Polynomial System.* CoRR, vol. abs/1104.4954, 2011. (Cited on page 10.)

[Emeliyanenko 2012] P. Emeliyanenko and M. Sagraloff. *On the Complexity of Solving a Bivariate Polynomial System.* In Proceedings of the 37th International Symposium on Symbolic and algebraic computation, ISSAC'12, pages 154–161, 2012. (Cited on pages 14 and 59.)

[Emiris 2008] Ioannis Z. Emiris, Athanasios Kakargias, Sylvain Pion, Monique Teillaud and Elias P. Tsigaridas. *Towards an Open Curved Kernel*, 2008. (Cited on page 127.)

[Emiris 2010] I. Z. Emiris, B. Mourrain and E. P. Tsigaridas. *The DMM bound: Multivariate (aggregate) separation bounds.* In S. Watt, editeur, Proceedings of the 35th International Symposium on Symbolic and Algebraic Computation - ISSAC'10, pages 243–250, Munich, Germany, July 2010. ACM. (Cited on page 27.)

[Everett 2007] H. Everett, S. Lazard, D. Lazard and M. Safey El Din. *The Voronoi diagram of three lines.* In Proceedings of the twenty-third annual symposium on Computational geometry, SCG '07, pages 255–264, New York, NY, USA, 2007. ACM. (Cited on page 1.)

[Fulton 1989] W. Fulton and R. Weiss. Algebraic curves: an introduction to algebraic geometry, volume 3. Addison-Wesley, 1989. (Cited on page 82.)

[Fulton 2008] W. Fulton. Algebraic curves: an introduction to algebraic geometry. 2008. Personal reprint made available by the author (http://www.math.lsa.umich.edu/ wfulton/CurveBook.pdf). (Cited on page 66.)

[Garloff 2000] J. Garloff and A. P. Smith. *Investigation of a Subdivision Based Algorithm for Solving Systems of Polynomial Equations*, 2000. (Cited on page 12.)

[Giusti 2001] M. Giusti, G. Lecerf and B. Salvy. *A Gröbner Free Alternative for Polynomial System Solving.* Journal of Complexity, vol. 17, no. 1, pages 154 – 211, 2001. (Cited on page 16.)

[Gonzalez-Vega 1989] L. Gonzalez-Vega, H. Lombardi, T. Recio and M.-F. Roy. *Sturm-Habicht Sequence.* In Proc. Int. Symp. on Symbolic and Algebraic Computation, pages 136–146, 1989. (Cited on page 128.)

[Gonzalez-Vega 1996] L. Gonzalez-Vega and M. El Kahoui. *An Improved Upper Complexity Bound for the Topology Computation of a Real Algebraic Plane Curve.* J. of Complexity, vol. 12, no. 4, pages 527–544, 1996. (Cited on pages 9, 10, 15, 17, 50, 51, 59, 61, 72, 76, 80, 81, 113, 151 and 152.)

[Gonzalez-Vega 2002] L. Gonzalez-Vega and I. Necula. *Efficient topology determination of implicitly defined algebraic plane curves.* Computer Aided Geometric Design, vol. 19, no. 9, 2002. (Cited on pages 17, 18, 131 and 140.)

[Hemmer 2010] Michael Hemmer. *Polynomial. In CGAL User and Reference Manual.* Rapport technique, CGAL Editorial Board, 3.6 edition, 2010. (Cited on page 129.)

[Hong 1996] H. Hong. *An efficient method for analyzing the topology of plane real algebraic curves.* In Selected papers presented at the international IMACS symposium on Symbolic computation, new trends and developments, pages 571–582, Amsterdam, The Netherlands, The Netherlands, 1996. Elsevier Science Publishers B. V. (Cited on page 18.)

[Kalkbrener 1993] M. Kalkbrener. *A generalized Euclidean algorithm for computing triangular representations of algebraic varieties.* J. Symb. Comput., vol. 15, no. 2, pages 143–167, February 1993. (Cited on page 15.)

[Kerber 2009a] M. Kerber. *Geometric Algorithms for Algebraic Curves and Surfaces.* PhD thesis, Universität des Saarlandes, Germany, 2009. (Cited on page 153.)

[Kerber 2009b] M. Kerber. *On the Complexity of Reliable Root Approximation.* In CASC'09: Proceedings of the 11th International Workshop on Computer Algebra in Scientific Computing, pages 155–167, Berlin, Heidelberg, 2009. Springer-Verlag. (Cited on page 14.)

[Kerber 2011] M. Kerber and M. Sagraloff. *A Worst-case Bound for Topology Computation of Algebraic Curves.* CoRR, vol. abs/1104.1510, 2011. (Cited on page 18.)

[Kerber 2012] M. Kerber and M. Sagraloff. *A worst-case bound for topology computation of algebraic curves.* J. Symb. Comput., vol. 47, no. 3, pages 239–258, 2012. (Cited on page 42.)

[Knuth 1971] D. Knuth. *The analysis of algorithm.* Act du congrès international des mathématiciens de 1970, vol. 3, pages 269–274, 1971. (Cited on page 25.)

[Labs 2009] O. Labs. *A List of Challenges for Real Algebraic Plane Curve Visualization Software.* In Nonlinear Computational Geometry, volume IMA 151, pages 137–164. Springer, 2009. (Cited on pages 131 and 140.)

[Lazard 1991] D. Lazard. *A new method for solving algebraic systems of positive dimension.* Discrete Appl. Math., vol. 33, pages 147–160, October 1991. (Cited on pages 15 and 50.)

[Lazard 1992] D. Lazard. *Solving Zero-Dimensional Algebraic Systems.* J. Symb. Comput., vol. 13, no. 2, pages 117–132, 1992. (Cited on page 15.)

[Lazard 2009] Sylvain Lazard, Luis Mariano Peñaranda and Elias P. Tsigaridas. *Univariate Algebraic Kernel and Application to Arrangements.* In Jan Vahrenhold, editeur, SEA, volume 5526 of *Lecture Notes in Computer Science*, pages 209–220. Springer, 2009. (Cited on pages 127, 128 and 130.)

[Lebreton 2013] R. Lebreton, E. Mehrabi and É. Schost. *On the complexity of solving bivariate systems: the case of non-singular solutions.* In Proceedings of the 38th International Symposium on Symbolic and Algebraic Computation, ISSAC '13, pages 251–258, New York, NY, USA, 2013. ACM. (Cited on page 15.)

[Li 2003] T.Y. Li. *Numerical Solution of Polynomial Systems by Homotopy Continuation Methods.* In Handbook of Numerical Analysis, volume 11 of *Handbook of Numerical Analysis*, pages 209 – 304. Elsevier, 2003. (Cited on page 12.)

[Li 2011] X. Li, M. Moreno Maza, R. Rasheed and É. Schost. *The modpn library: Bringing fast polynomial arithmetic into Maple.* J. Symb. Comput., vol. 46, no. 7, pages 841–858, 2011. (Cited on pages 15, 50, 51, 76, 80, 81, 82 and 132.)

[Lickteig 2001] T. Lickteig and M.-F. Roy. *Sylvester-Habicht Sequences and Fast Cauchy Index Computation.* J. Symb. Comput., vol. 31, no. 3, pages 315–341, 2001. (Cited on pages 13, 25, 26, 34, 112, 114 and 115.)

[Lombardi 1990] H. Lombardi. *Sous-Résultant, suite de Sturm, spécialisation.* PhD thesis, Université de Franche Comté, 1990. (Cited on page 114.)

[Lorensen 1987] W. E. Lorensen and H. E. Cline. *Marching cubes: A high resolution 3D surface construction algorithm.* SIGGRAPH Comput. Graph., vol. 21, pages 163–169, August 1987. (Cited on page 18.)

[Mehlhorn 2008] K. Mehlhorn and M. Sagraloff. *A Deterministic Descartes Algorithm for Real Polynomials*, 2008. (Cited on pages 5 and 127.)

[Mehlhorn 2013] K. Mehlhorn, M. Sagraloff and P. Wang. *From Approximate Factorization to Root Isolation with Application to Cylindrical Algebraic Decomposition.* CoRR, vol. abs/1301.4870, 2013. (Cited on pages 27 and 153.)

[Mignotte 1989] M. Mignotte. Mathématiques pour le calcul formel. Presses Universitaires de France, 1989. (Cited on page 69.)

[Moore 2009] R.E. Moore, M.J. Cloud and R.B. Kearfott. Introduction to interval analysis. SIAM e-books. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2009. (Cited on page 12.)

[Mourrain 2009] B. Mourrain and J.P. Pavone. *Subdivision methods for solving polynomial equations.* J. Symb. Comput., vol. 44, no. 3, pages 292–306, 2009. Polynomial System Solving in honor of Daniel Lazard. (Cited on page 12.)

[MPFI] MPFI. *MPFI: multiple precision interval arithmetic library.* `http://mpfr.org`. (Cited on page 130.)

[MPFR] MPFR. *MPFR: a library for multiple-precision floating-point computations.* `http://mpfr.org`. (Cited on page 130.)

[Neumaier 1990] A. Neumaier. Interval methods for systems of equations. Cambridge Middle East Library. Cambridge University Press, 1990. (Cited on page 12.)

[Niang Diatta 2008] D. Niang Diatta, B. Mourrain and O. Ruatta. *On the computation of the topology of a non-reduced implicit space curve.* In ISSAC'08: Proceedings of the twenty-first International Symposium on Symbolic and algebraic computation, pages 47–54, New York, NY, USA, 2008. ACM. (Cited on pages 75, 102 and 153.)

[Peñaranda. 2010] L. Peñaranda. *Géometrie algorithmique non linéaire et courbes algébriques planaires.* Ph.d. thesis, Universite Nancy 2, France, 2010. (Cited on page 140.)

[Plantinga 2004] S. Plantinga and G. Vegter. *Isotopic approximation of implicit curves and surfaces.* In SGP'04: Eurographics/ACM SIGGRAPH Symposium on Geometry Processing, pages 245–254, 2004. (Cited on page 18.)

[Reischert 1997] D. Reischert. *Asymptotically fast computation of subresultants.* In Proceedings of the 1997 International Symposium on Symbolic and algebraic computation, ISSAC'97, pages 233–240, New York, NY, USA, 1997. ACM. (Cited on pages 34 and 112.)

[Renegar 1989] J. Renegar. *On the worst-case arithmetic complexity of approximating zeros of systems of polynomials.* SIAM J. Comput., vol. 18, no. 2, pages 350–370, April 1989. (Cited on page 16.)

[Rheinboldt 1998] W.C. Rheinboldt. Methods for solving systems of nonlinear equations. CBMS-NSF Regional Conference Series in Applied Mathematics. Society for Industrial and Applied Mathematics, 1998. (Cited on page 12.)

[Rouillier 1999] F. Rouillier. *Solving zero-dimensional systems through the rational univariate representation.* J. of Applicable Algebra in Engineering, Communication and Computing, vol. 9, no. 5, pages 433–461, 1999. (Cited on pages 4, 9, 16, 17, 21, 59, 61, 65, 75, 77, 80, 84, 88 and 100.)

[Rouillier 2003] F. Rouillier and P. Zimmermann. *Efficient Isolation of Polynomial Real Roots.* J. of Computational and Applied Mathematics, vol. 162, no. 1, pages 33–50, 2003. (Cited on pages 5, 127 and 137.)

[Roy 1996] M.-F. Roy. *Basic algorithms in real algebraic geometry and their complexity : from Strum theorem to the existential theory of reals.* Lectures on Real Geometry in memoriam of Mario Raimondo, Gruyter Expositions in Mathematics., vol. 23, pages 1–67, 1996. (Cited on page 114.)

[Rump 1983] S.M. Rump. Solving algebraic problems with high accuracy. 1983. (Cited on page 12.)

[Sagraloff 2012] M. Sagraloff. *When Newton meets Descartes: A Simple and Fast Algorithm to Isolate the Real Roots of a Polynomial.* In Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation, ISSAC'12, pages 297–304, 2012. (Cited on page 11.)

[Schost 2001] É. Schost. *Sur la Résolution des Systèmes Polynomiaux à Paramètres.* PhD thesis, Ecole Polytechnique, France, 2001. (Cited on page 62.)

[Schönhage 1982] A. Schönhage. The fundamental theorem of algebra in terms of computational complexity. Manuscript. Department of Mathematics, University of Tübingen. Updated 2004., 1982. (Cited on page 25.)

[Seidel 2005] R. Seidel and N. Wolpert. *On the exact computation of the topology of real algebraic curves.* In Proc 21st ACM Symposium on Computational Geometry, pages 107–115, 2005. (Cited on pages 13 and 17.)

[Sherbrooke 1993] E. C. Sherbrooke and N. M. Patrikalakis. *Computation of the solutions of nonlinear polynomial systems.* Computer Aided Geometric Design, vol. 10, no. 5, pages 379 – 405, 1993. (Cited on page 12.)

[Strzebonski 2012] A. W. Strzebonski and E. P. Tsigaridas. *Univariate real root isolation in multiple extension fields.* In Proceedings of the 37th International Symposium on Symbolic and Algebraic Computation - ISSAC, pages 343–350, 2012. (Cited on page 16.)

[Teissier 1973] B. Teissier. *Cycles évanescents, sections planes et conditions de Whitney. (French).* In Singularités à Cargèse (Rencontre Singularités Géom. Anal., Inst. Études Sci., Cargèse, 1972), numéro 7–8 de Asterisque, pages 285–362. Soc. Math. France, Paris, 1973. (Cited on page 119.)

[Torbjörn Granlund 2002] et al. Torbjörn Granlund. *GNU Multiple Precision Arithmetic Library 4.1.2*, December 2002. `http://swox.com/gmp/`. (Cited on page 129.)

[Van der Waerden 1930] B.-L. Van der Waerden. Moderne algebra 1. Springer, Berlin, 1930. (Cited on page 42.)

[Verschelde 2010] J. Verschelde. *Polynomial Homotopy Continuation with PHCpack*, 2010. (Cited on page 12.)

[von zur Gathen 2003] J. von zur Gathen and J. Gerhard. Modern computer algebra. Cambridge Univ. Press, Cambridge, U.K., 2nd édition, 2003. (Cited on pages 19, 22, 23, 24, 34, 35, 36, 38, 39, 55, 56, 86, 90, 97, 100, 116 and 118.)

[Wang 2001] D. Wang. Elimination methods. Texts & Monographs in Symbolic Computation. Springer, 2001. (Cited on page 15.)

[Yap 2000] C. K. Yap. Fundamental problems of algorithmic algebra. Oxford University Press, Oxford-New York, 2000. (Cited on pages 6, 19, 25, 26, 27, 34, 35, 36, 38, 39, 56, 90, 94 and 112.)