EFFICIENT POLYNOMIAL ROOT ISOLATION APPLIED TO COMPUTATIONAL GEOMETRY

An Undergraduate Research Scholars Thesis

by

JORDAN LAMKIN

Submitted to the Undergraduate Research Scholars program at
Texas A&M University
in partial fulfillment of the requirements for the designation as an

UNDERGRADUATE RESEARCH SCHOLAR

Approved by Research Advisor: Dr. John Keyser

November 2019

Major: Computer Science

TABLE OF CONTENTS

ABSTRACT

Efficient Polynomial Root Isolation Applied to Computational Geometry

Jordan Lamkin
Department of Computer Science
Texas A&M University

Research Advisor: Dr. John Keyser
Department of Computer Science
Texas A&M University

Over the last several years, the field of polynomial root isolation has been rapidly improving, but the computational geometry applications have been somewhat unexplored. Here, we have an implementation of a curve intersection engine that showcases the current state-of-the-art in root isolation. The engine is capable of taking two implicitly defined curves and locating their intersection points within some required accuracy. From this work, we can clearly see that root isolation is no longer a significant speed issue in computational geometry. The next issue is really speed of the resultant computation used for variable elimination.

1. INTRODUCTION

In the field of computational geometry, and in particular CAD-style modelling applications, numerical error, if present at all, can propogate rapidly and can cause incorrect branching, leading to wrong output or to program crashes. [1] This kind of issue can be solved by using exact computation and storing symbolic representations of points, curves and surfaces.

1.1 Motivation

While exact computation libraries exist, current libraries run too slowly to be useful for working with real problems in real time. For this reason, most commercial CAD systems use floating point approximations and work to minimize the cases of incorrect results and crashes. We would like to be able to use exact computation to construct a CAD system free from these issues.

The current biggest slowdown in exact computation based CAD systems is in the polynomial root isolation algorithms to which many key computations reduce. These include isolation, matching and sorting of intersection points between curves or curved surfaces [1], which in turn can be used to achieve almost any operation on boundary-surface-based representations of solids [2].

A variety of techniques have been used to find roots of polynomials in computational geometry. These include algorithms based on resultants of polynomials, Sturm sequences, worst-case bit-length estimates, Grobner Basis, as well as algorithms proposed explicitly for this purpose in MAPC. [1, 3]

However, while root finding is a well-studied topic, it remains a computationally costly problem in these contexts, and thus an open area of research. In particular, the algorithm described in [1] uses a polynomial root isolation algorithm based on Sturm's theorem.

1.2 Hypothesis

In MAPC, polynomial root isolation accounts for upwards of 90% of the total time involved in curve-curve intersection. Over the last several years however, there have been drastic improvements to the state-of-the-art in polynomial root isolation algorithms. We hypothesize that by using more recent, much more efficient root isolation algorithms, we can reduce the run time of an exact computation based curve intersection engine to within tolerable levels.

2. ROOT ISOLATION ALGORITHMS

We now turn our attention to the problem of finding a suitable univariate polynomial root isolation algorithm. Polynomial root isolation is an old problem with a multitude of approaches, and while many more algorithms than those discussed here have been used, we will only consider some of the more recent ones.

In general, the problem is split into two parts: First, an arbitrary polynomial must be reduced to a squarefree polynomial. Second, the roots of the squarefree polynomial are isolated using the assumption that the polynomial is squarefree. The first step is historically much less challenging as it can be done simply by dividing the polynomial by its gcd with its derivative. As such, it is the second step in this process that most polynomial root isolation algorithms truly focus on.

Algorithms which isolate roots of squarefree polynomials are generally based on one of either Sturm's theorem or Descartes' rule of signs. The general approach in these algorithms is to continually split a target interval until each subinterval is guaranteed to contain at most one root, and to discard the interval if it does not contain a root.

2.1 Mathematical Background

2.1.1 Sturm Sequences

The Sturm sequence $P_0, P_1, ..., P_k$ of a polynomial P is given by $P_0 = P$, $P_1 = P'$ and $P_{n+2} = -\mathrm{rem}(P_n, P_{n+1})$ where P' denotes the derivative of P and $\mathrm{rem}(P, Q)$ denotes the polynomial remainder of P divided by Q. Since the degree of $\mathrm{rem}(P, Q)$ is necessarily less than that of Q when Q is nonconstant and $\mathrm{rem}(P, Q) = 0$ when Q is constant, there can only be finitely many nonzero elements in this sequence. So we take P_k to be the last nonzero element.

2.1.2 Sturm's Theorem and Descartes' Rule of Signs

Sturm's theorem relates to Sturm sequences, stating that for a given $x_1, x_2 \in \mathbb{R}$, the difference in the number of sign variations between $P_0(x_1), P_1(x_1), ..., P_k(x_1)$ and $P_0(x_2), P_1(x_2), ..., P_k(x_2)$ is equal to the number of roots in the interval (x_1, x_2). This allows a simple way of determining the number of roots in an interval precisely.

Descartes' rule of signs states that the number of sign variations in the coefficients of a polynomial is greater than or equal to the number of positive roots. Similar to Sturm's theorem, Descartes' rule of signs provides a way of measuring roots in an interval. Unlike Sturm's theorem however, it can only give an upper bound and only works on the interval $(0, \infty)$. One can extend it to work on any interval by applying appropriate Taylor shifts or Möbius transformations.

2.2 Sturm Sequence Root Isolation Algorithm

Sturm's theorem suggests a simple root isolation algorithm, which is used by MAPC and the like to solve univariate polynomial equations. This algorithm consists of subdividing intervals with more than one root recursively until all intervals contain at most one root.

This algorithm is limited by two main factors:

1. The speed at which the Sturm sequence can be computed.

2. The number of splittings necessary to distinguish roots.

In practice, it turns out that computing the Sturm sequence is computationally expensive enough that it renders the algorithm impractical for degrees upward of around $n = 80$.

2.3 Rule of Signs Isolation Algorithms

Since the creation of MAPC, new work has proven [4] the termination of a similar algorithm using Descartes' rule of signs, the key difference being that some intervals may

turn out not to contain any roots once subdivided.

Since the coefficients of the polynomial are readily available, this algorithm's efficiency is limited only by:

1. The speed at which the Möbius transformation can be computed.

2. The upper bound on the number of subdivisions necessary to distinguish roots.

In practice, this algorithm works extremely well, but has very bad convergence rates for some edge cases. This has motivated further research to lower the upper bound.

2.3.1 ANEWDSC

Recent work in root isolation has improved this upper bound significantly. The best currently known algorithm, called ANEWDSC [5], is based on the bisection algorithm using Descartes' rule of signs, making two key improvements: allowing arbitrary precision real coefficients, and using a Newton test to achieve quadratic root convergence (rather than linear). This algorithm has a running time of $\mathcal{O}\left(n^3 + n^2 L\right)$ and practically speaking, it is able to efficiently handle degrees above $n = 1000$ consistently.

This algorithm has been implemented in an open source C library called SLV [6], and this forms the backbone of our library.

2.3.2 Application in Computational Geometry

While much work has been done recently in constructing algorithms such as ANEWDSC, this work has not yet been applied in the context of exact computational geometry. The most recent work in this field is still using outdated Sturm sequence techniques.

3. IMPLEMENTATION

3.1 Capabilities

We provide a library capable of testing our hypothesis that a fast polynomial root isolation algorithm may be used to create a usably fast curve intersection algorithm. Our library provides

- A C++ wrapper around SLV, giving us a fast polynomial root isolation algorithm.
- A simple infrastructure for eliminating variables from systems of two variable polynomials.
- The curve-curve intersection method whose overall running time and profiling we are interested in.

This library is constructed with two goals in mind: testing the above hypothesis, and providing a foundation upon which a future exact computation based CAD system may be written.

This library may be found at:

`https://github.com/JDLamkin/Curve-Intersect`

3.1.1 SLV Wrapper

Our library is built on top of the SLV fast root-finding library. SLV gives a C interface for finding roots of polynomials with integer coefficients in $\mathcal{O}\left(n^3 + n^2 L\right)$ time, which is a significant theoretical improvement over the Sturm sequence approach. However, it is awkward to work with on the large scale required for implementing a CAD system, so we first construct a C++ wrapper around SLV to vastly simplify the process of finding

and refining root intervals and to interface directly with the C++ version of the big integer library GMP.

We then extend this with specialized classes for one and two dimensional rational-coefficient polynomials for general use. These are used as building blocks to construct the complex algorithms used for curve-curve intersection.

3.1.2 Resultant Computation

Now, loosely following the curve-curve intersection algorithm from MAPC [1], we must implement an algorithm for Sylvester resultants. This will be used exactly twice per intersection to find the set of possible x-locations and possible y-locations of roots.

However, as we will see below, the computation of these two resultants is now by far the dominant slowdown in our curve-curve intersection algorithm, so in an effort to optimize this process, we use the Bariess determinant algorithm (which can be shown to work for matrices whose entries are commutative rings [7]) on the Sylvester matrix constructed from our two curves.

3.1.3 Point Testing

Finally, we must implement some means to test a pair of x and y intervals to discover whether they contain an intersection point. Again by the algorithm from MAPC, we determine the number and order of crossings of the two curves with the boundary of the region in question using once again our root isolation algorithm.

3.2 Limitations

Unfortunately, our system still has three fairly major limitations that cause it to fail in many cases:

- SLV itself does not yet support roots at dyadic numbers (integers divided by a power of 2).

- SLV makes no attempt to support non-squarefree polynomials and our library does not yet fix this.
- Our library does not yet support tangential intersections of curves.

This means in general that curves which pass through dyadic lattice points or which have inflection points with dyadic limits can cause potential root regions to be misidentified.

4. RESULTS AND CONCLUSIONS

4.1 Sample Results

Figures 4.1, 4.2, and 4.3 give three examples of our library's behavior on various inputs. In these figures we use a green rectangle to indicate the unrefined root region and a black dot to represent the root found (refined to within < 1 pixel of tolerance). We can see from the figures that all roots are found successfully.

These equations are intentionally chosen not to pass through dyadic lattice points or to contain inflection points with dyadic limits as per the limitations in the previous section.

4.2 Timing Experiments

4.2.1 Test Case Considerations

In general, the efficiency of root isolation algorithms is significantly impacted by root clustering, however since SLV has been shown [6] to work equally well in all test cases, we turn our attention instead to bitlength and to polynomial degree.

In the experiments described below, we use bidegree n and bitlength L to describe a polynomial whose terms are given by $Ax^{k_1}y^{k_2}$ where $-2^L < A < 2^L$ and $0 \leq k_1, k_2 \leq n$. Each such A is uniformly randomly chosen in that range. The results of each experiment are averaged over 10 such randomly chosen polynomials.

These experiments are done on an Intel® Core™ i7-5500U Processor, and the resulting times are listed in seconds.

4.2.2 Experiments

Figure 4.4 provides a simple timing test of the curve intersection function over this kind of random data, where figure 4.5 shows how the $L = 30$ case breaks down across root isolation, resultant finding, and the remainder of the algorithm.

$$C_1 : -16x^2 + 24x - 16y^2 + 16y + 3 = 0$$
$$C_2 : 2x^4 - 8x^3 + 4x^2y^2 - 8x^2y + 9x^2 - 8xy^2 + 12x + 2y^4 - 8y^3 + 9y^2 + 12y - 18 = 0$$

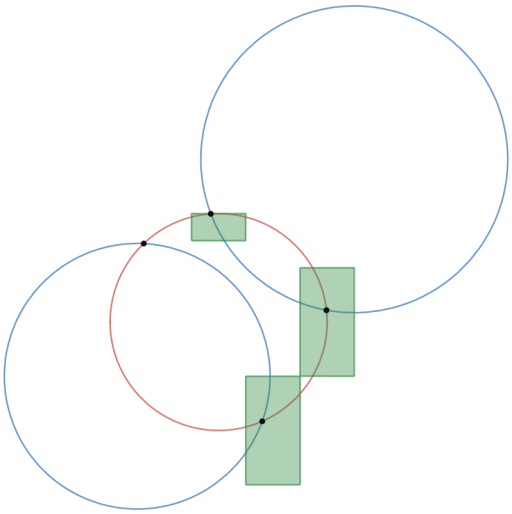

Figure 4.1: First example intersection

$$C_1 : x^2 + 4y^2 - 3 = 0$$
$$C_2 : x^3 + 2xy^2 - 5xy + y^3 - 2 = 0$$

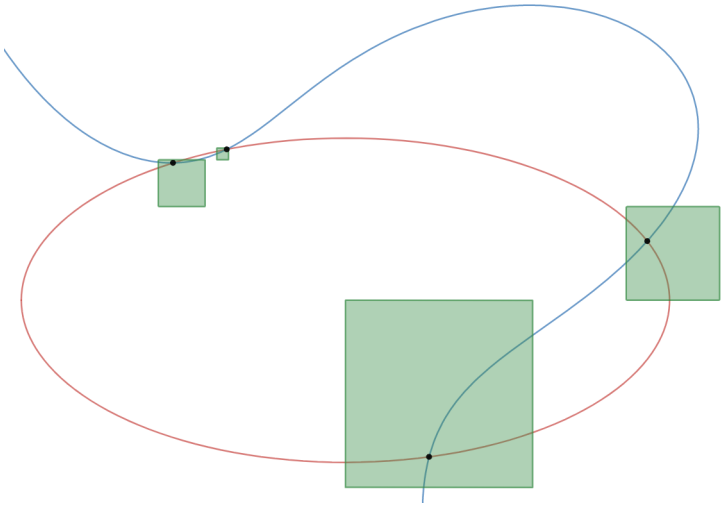

Figure 4.2: Second example intersection

$$C_1 : 14x^2 + 15y^2 - 1 = 0$$
$$C_2 : 160x^4 - 1200x^3y + 1930x^2y^2 - 170x^2 + 1200xy^3 + 160y^4 - 170y^2 + 9 = 0$$

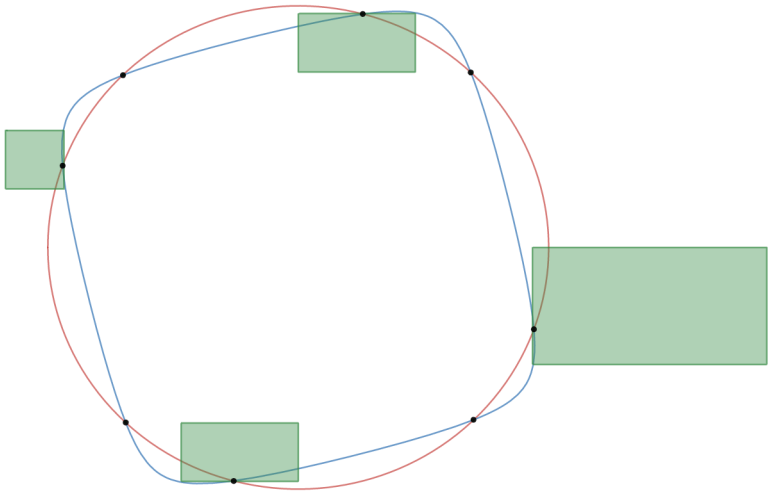

Figure 4.3: Third example intersection

	$L = 10$	$L = 30$	$L = 100$	$L = 300$
$n = 2$	0.005	0.005	0.006	0.007
$n = 3$	0.029	0.015	0.020	0.021
$n = 4$	0.072	0.068	0.078	0.130
$n = 5$	0.244	0.223	0.294	0.653
$n = 6$	0.746	0.719	1.022	2.707
$n = 7$	2.889	1.919	2.926	10.140

Figure 4.4: Total Time (in seconds) by Curve Degree and Bitlength

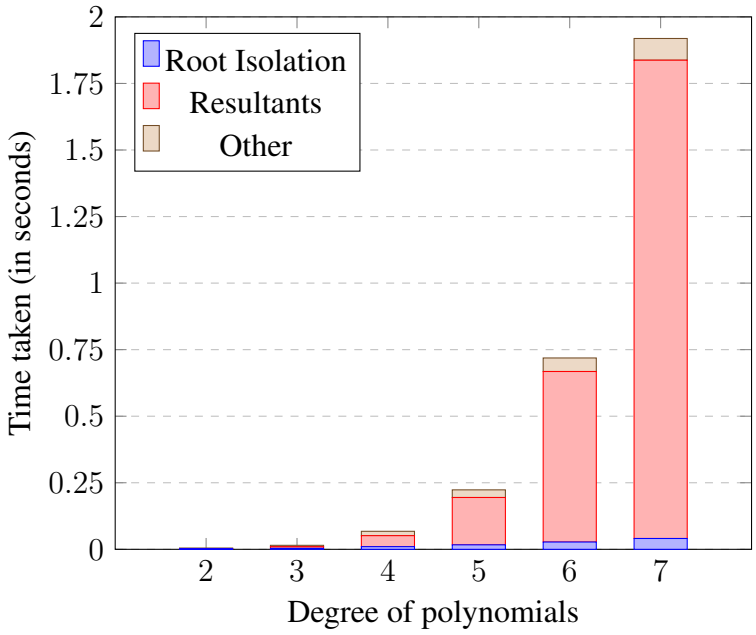

Figure 4.5: Timing Breakdown of Curve-Curve Intersection by Curve Degree

4.3 Conclusions

Based on the figures above, we note several things in support of our hypothesis. First, we note that for bitlengths under 100, the bitlength has very little effect on the time, and in such cases, we see that our curve intersection engine runs in under a second for input degrees ≤ 6.

We further note that the vast majority of this time comes from the resultant algorithm used for variable elimination. This will be discussed in more detail below.

4.4 Future Work

We believe our library to be a good start toward an efficient exact computation based CAD system, but there is still a lot of work to be done.

4.4.1 Efficient Resultant Computation

As it stands, it would appear that our library is just fast enough to be practical, but it still does have significant slowdown for variable elimination. In this system, >75% of the total run time is accounted for by the computation of resultants used for elimination of variables for input curves of degree 5 or more.

In an attempt to optimize this part of the algorithm, we have used an efficient matrix determinant algorithm to compute the Sylvester resultant, but this algorithm is still not efficient enough to be useful for larger input sizes.

We believe that this motivates work toward more efficient resultant computation.

4.4.2 Edge Case Implementation

The curve intersection engine as it stands is not robust. There are a number of edge cases that are simply not accounted for becuase doing so would add implementation complexity and project time was constrained. Among these edge cases are:

- SLV sometimes returns bad output for polynomials with dyadic roots. We suspect a simple change to SLV could fix this.
- SLV does not support non-squarefree polynomials, but no check is made prior to calling.
- Potential roots containing tangent intersections of curves are not supported. This could possibly be fixed by calling for refinement on the ambiguous case until the minimum possible root separation is reached.
- Potential root regions with tangent intersections to curves are not supported. This is dependent on the support of both dyadic roots and non-squarefree polynomials and can be corrected by checking for and removing tangent boundary points.

4.4.3 Full 3D CAD System

Finally, while curve-curve intersection is an important algorithm at the core of a CAD system, it still remains to be shown that exact computation can be used to make an efficient, real time CAD system. We would like to see our library used as the core of a full 3D CAD library.

REFERENCES

[1] J. C. Keyser, *Exact boundary evaluation for curved solids*. The University of North Carolina at Chapel Hill, 2000.

[2] J. Keyser, T. Culver, M. Foskey, S. Krishnan, and D. Manocha, "Esolid—a system for exact boundary evaluation," *Computer-Aided Design*, vol. 36, no. 2, pp. 175–193, 2004.

[3] J. Keyser, T. Culver, D. Manocha, and S. Krishnan, "Mapc: A library for efficient and exact manipulation of algebraic points and curves," in *Proceedings of the fifteenth annual symposium on Computational geometry*, pp. 360–369, ACM, 1999.

[4] A. Eigenwillig, "Real root isolation for exact and approximate polynomials using descartes' rule of signs," 2008.

[5] A. Kobel, F. Rouillier, and M. Sagraloff, "Computing real roots of real polynomials ... and now for real!," in *Proceedings of the ACM on International Symposium on Symbolic and Algebraic Computation*, ISSAC '16, (New York, NY, USA), pp. 303–310, ACM, 2016.

[6] E. Tsigaridas, "Slv: a software for real root isolation," *ACM Communications in Computer Algebra*, vol. 50, no. 3, pp. 117–120, 2016.

[7] G. I. Malashonok, "Algorithms for the solution of systems of linear equations in commutative rings," in *Effective Methods in Algebraic Geometry*, pp. 289–298, Springer, 1991.