

# Constructing Cylindrical Algebraic Decompositions of the Plane Quickly

Christopher W. Brown

January 14, 2002

## Abstract

This paper presents a method for speeding up the construction of Cylindrical Algebraic Decompositions (CADs) of 2-dimensional space, in which exact computations over the highest degree algebraic extensions are replaced with numerical calculations. The method uses information gathered during the CAD construction process to produce guaranteed correct results from these approximate computations. As a result, a CAD for a set of bivariate polynomials with irreducible discriminants and resultants can be constructed without any computations over algebraic extensions.

The method has been implemented, and the paper reports the results of many experimental trials. These results show not only that the new method speeds up CAD construction for problems that are within the reach of the old method, but also that many 2D CAD construction problems that cannot be performed in a reasonable amount of time with the old method can be completed quite quickly with these improvements.

## 1 Introduction

A set  $A$  of polynomials in  $x$  and  $y$  partitions  $\mathbb{R}^2$  into points, curve segments, and open regions in a natural way — namely into the maximal connected regions in which elements of  $A$  have invariant sign. Cylindrical Algebraic Decomposition (CAD) [Col75, CH91], especially with the addition of “adjacency computation” [ACM84b], is a powerful tool for analyzing such partitions. It can be used to determine the satisfiability of constraints involving elements of  $A$  and to simplify large boolean combinations of such constraints, it can provide complete topological descriptions of any of the regions in the partition — including the real algebraic curves defined by the elements of  $A$ , and it can be used to produce reliable plots of any region within the partition.

The most computationally expensive part of 2-dimensional CAD construction is *lifting* (also called *stack construction*). Lifting over a point  $\alpha \in \mathbb{R}$  involves analyzing the arrangement of the real roots of the set of univariate polynomials in  $y$  that results from substituting  $\alpha$  for  $x$  in each of the elements of  $A$ . The  $\alpha$ 's over which we lift are potential  $x$ -coordinates of singular points, vertical tangencies, intersection points and vertical asymptotes, which are typically algebraic numbers — the roots of discriminants, resultants, and leading coefficients of the elements of  $A$ . The CAD algorithm lifts over  $\alpha$  by:

1. substituting  $\alpha$  into the elements of  $A$ , which yields a set  $A' \subset \mathbb{Q}(\alpha)[y]$ ,
2. computing  $B$ , a squarefree basis for  $A'$ , which involves computing gcd's of polynomials with coefficients in  $\mathbb{Q}(\alpha)$ , and
3. computing an ordered list of isolating intervals for the real roots of the elements of  $B$ .

The exact substitution and squarefree basis computation for the algebraic number  $\alpha$  is quite expensive, both theoretically and in practice. This could also have been said of the root isolation phase, but the use of the interval Descartes method as described in [CJK] has dramatically sped-up this portion of the computation, so that it is, in practice, negligible in comparison to substitution and squarefree basis computation. In any event, these steps are most expensive when the minimal polynomial of  $\alpha$  is of high degree and has large coefficients, which arises when one of the discriminants or resultants is irreducible, or has a multiplicity-one factor that is exceptionally large.

This paper shows that in the cases that are especially bad for exact substitution and squarefree basis computation, we can usually get the necessary information about the roots of elements of  $A$  at  $x = \alpha$  without using exact substitution or squarefree basis computation. This gives rise to an improved lifting method for 2D CAD's. The idea underlying this new method is that the elements of  $A$  are actually very well-behaved over simple roots of resultants and discriminants, so that symbolic methods aren't really needed to get exact information. Moreover, the simple roots typically generate higher-degree extension fields, so that the symbolic methods are slowest where they're not needed! On the other hand, higher-multiplicity roots of a discriminant may indicate singularities, and higher-multiplicity roots of a resultant may indicate a point of mutual tangency between two curves, and these seem to require symbolic methods to determine exact information. However, the higher-multiplicity roots typically generate lower-degree extensions, so that symbolic methods are fastest where they're really needed!

The effect of the improved lifting method is that many stack constructions can proceed without any exact substitutions or squarefree basis computations. For some stack constructions this may not be possible, but in these cases we simply

fall back on the old lifting method. Experimental data reported in this paper considers several different kinds of input polynomial sets, and speedup factors resulting from the new lifting method range up into the thousands. There are several ways in which the results of this paper are potentially important:

- Simplification of large boolean combinations of bivariate polynomial sign conditions, as described in [Bro01], will be dramatically improved by the addition of these methods. This problem is important if one is to make effective use of, for example, the Redlog system for quantifier elimination [DS96].
- It will be possible to analyze the topology of much larger degree real algebraic curves using CAD and adjacency computation [ACM84b].
- These improvements make CAD-based methods for manipulating algebraic points and curves competitive with and perhaps superior to, for example, the special purpose methods devised in [KCMK00]. This may make CAD an attractive alternative for exact computations in Computer-Aided Design.
- The same ideas that are proposed in this paper for 2D CAD construction may, perhaps, be applied to 3D CAD construction as well. There the CAD algorithm starts to deal with towers of extensions, so in addition to avoiding exact substitution and squarefree basis computations, we may be able to avoid primitive element computations, and we may be able to avoid *very* large algebraic extensions.

The remainder of this paper is organized as follows: Section 2 describes the usual method of CAD construction in  $\mathbb{R}^2$ . Section 3 describes the improved lifting method. Section 4 describes our test implementation of the improved lifting method as part of the QEPCAD program for CAD construction and quantifier elimination. Experimental data gathered from our test implementation is reported in Section 5. And finally, Section 6 provides some conclusions and areas for future work.

## 2 CADs of the plane

This section describes what a cylindrical algebraic decomposition of  $\mathbb{R}^2$  is, and how one may be constructed. A detailed description may be found in [ACM84a].

## 2.1 Cylindrical Algebraic Decomposition

Any set  $P$  of polynomials in  $\mathbb{R}[x, y]$  defines a *natural algebraic decomposition* of  $\mathbb{R}^2$  into the maximal connected regions in which the elements of  $P$  have invariant sign. If the sets comprising such a decomposition are also cylindrically arranged, meaning that the projections of any pair of sets onto  $\mathbb{R}^1$  are either identical or disjoint, then the natural algebraic decomposition defined by  $P$  is a *cylindrical algebraic decomposition*. Such a set  $P$  is called a *projection factor set*.

It is useful to separate the elements of a projection factor set  $P$  into two sets,  $P_1$ , which consists of all projection factors of degree 0 in  $y$ , and  $P_2$ , which consists of all projection factors of positive degree in  $y$ .  $P_1$  defines a natural algebraic decomposition of  $\mathbb{R}^1$  into open intervals and single points. This decomposition is actually a CAD of  $\mathbb{R}^1$ , which we call the *induced CAD* of  $\mathbb{R}^1$ . The *cells*, as the sets comprising a CAD are called, of the induced CAD are actually the projections of cells from the CAD defined by  $P$ .

The projection factor set  $P$  provides an implicit representation of a CAD. The *lifting* or *stack construction* process constructs an explicit representation — a CAD data structure — from the set  $P$ . This proceeds in two steps: first we construct an explicit representation of the cells from the induced CAD of 1-space, then for each cell  $c$  in the induced CAD we construct an explicit representation of those cells from the CAD defined by  $P$  that project down onto  $c$  (these cells are *stacked* above  $c$ ). Each cell is represented by a *sample point*, which is simply a point from the cell, the signs of each of the projection factors in the cell and, in the case of a cell from the induced CAD, a list of the cells that are stacked above it.

To construct the induced CAD we compute an ordered list  $\alpha_1 < \alpha_2 < \dots < \alpha_n$  of all real roots of elements of  $P_1$ . We then chose rational points  $r_1, \dots, r_{n+1}$  such that  $r_1 < \alpha_1 < r_2 < \dots < \alpha_n < r_{n+1}$ . Finally, we construct the list  $(c_1, \dots, c_{2n+1})$  of cells comprising the induced CAD, where  $r_1$  is the sample point of  $c_1$ ,  $\alpha_1$  is the sample point of  $c_2$ , etc.

If  $c$  is a cell from the induced CAD with sample point  $\beta$ , we construct a list of the cells stacked above  $c$  by simply substituting  $x = \beta$  in the elements of  $P_2$  yielding a set  $P_c \subset \mathbb{Q}(\beta)[y]$ , which we then treat as a projection factor set for CAD of 1-space. Thus, we compute an ordered list  $\alpha_1 < \alpha_2 < \dots < \alpha_n$  of all real roots of elements of  $P_c$ . We then chose rational points  $r_1, \dots, r_{n+1}$  such that  $r_1 < \alpha_1 < r_2 < \dots < \alpha_n < r_{n+1}$ . Finally, we construct the list  $(c_1, \dots, c_{2n+1})$  of cells, where  $(\beta, r_1)$  is the sample point of  $c_1$ ,  $(\beta, \alpha_1)$  is the sample point of  $c_2$ , etc. Figure 1 illustrates the correspondence between the decomposition of the plane and the data structure.

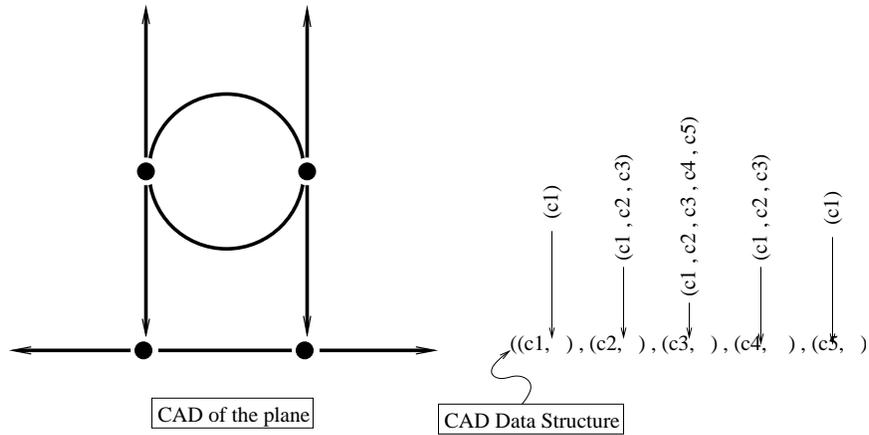


Figure 1: A CAD of the plane and the associated CAD data structure (minus the sample points).

## 2.2 Projection

Let  $A$  be a given set of polynomials in  $\mathbb{Z}[x, y]$ . Constructing a CAD for  $A$  means first producing a projection factor set  $P$  that contains all irreducible factors of  $A$  (the *projection* step), and then constructing a CAD data structure from  $P$  (the *lifting* step). With the CAD data structure, many questions concerning the elements of  $A$  are easily answered. For example, we can easily determine the satisfiability of any boolean combination of sign conditions involving elements of  $A$  — the cells of the CAD give all of the possible combinations of sign conditions involving projection factors. Additionally, it is often important to know which cells are adjacent to one another. This can be computed quite quickly for CAD's of  $\mathbb{R}^2$  using, for example, the method described in [ACM84b]. With adjacency information, one may analyze the topology of sets defined by the elements of  $A$ , which may be used to produce reliable plots [Arn83].

Let  $A_C$  be the set of all leading coefficients of elements of  $A$ ,  $A_D$  be the set of all discriminants eliminating  $y$  of elements of  $A$ , and  $A_R$  be the set of all resultants eliminating  $y$  of pairs of elements of  $A$ . We define  $P$ , the projection factor set constructed from  $A$ , as the set of all irreducible factors of the elements of  $A \cup A_C \cup A_D \cup A_R$ .

### 2.3 Real root isolation and lifting

The most time consuming part of constructing CAD's of  $\mathbb{R}^2$  is lifting over algebraic sample points. Let  $\alpha$  be an algebraic number with minimal polynomial  $M$ . The current method of lifting over  $\alpha$  is the following:

1. Substitute  $\alpha$  into  $P_2$ , resulting in the set  $P'_2$ .
2. Compute  $B$ , a squarefree basis for the elements of  $P'_2$ . This involves repeated gcd calculations in the domain  $\mathbb{Q}(\alpha)[y]$ , which are quite costly.
3. Compute an ordered list of non-overlapping *isolating intervals* for the real roots of the elements of  $B$ .

The third step can be performed with a version of the coefficient sign variation method (often called the Descartes method) [CA76], for polynomials with algebraic number coefficients. However, it can usually be done much more quickly with the *interval Descartes method* [JK97], which uses floating point numbers and interval arithmetic. The interval Descartes method of polynomial real root isolation embeds the given polynomial in an interval polynomial, i.e. each coefficient of the interval polynomial is a floating point interval containing the associated coefficient of the given polynomial. Let  $p^*$  be an interval polynomial containing some polynomial  $p$ . Given  $p^*$ , the interval Descartes method will return either *FAIL*, or a list of intervals that are isolating intervals for all the real roots of any polynomial that may be embedded in  $p^*$ . In particular, if *FAIL* is not returned, the interval Descartes method returns a list of isolating intervals for all real roots of  $p$ .

The interval Descartes method may fail for two basic reasons: overflow/underflow in the floating point arithmetic, or an inability to determine the sign of a number represented as an interval because that interval straddles zero. Higher floating point precision will always address the first cause of failure, but not necessarily the second. For example, if some non-squarefree polynomial is embeddable in  $p^*$ , then the algorithm can't possibly return anything but *FAIL*. Otherwise, no matter what intervals it returned, there would always have to be one interval that contained one root for some polynomials embeddable in  $p^*$ , and zero or two roots for some others. In the execution of the interval Descartes method, this manifests as an interval straddling zero.

The intriguing thing about the interval Descartes method is how often it doesn't return *FAIL*. Using it in place of the exact symbolic version of the Descartes method often considerably reduces the time for root isolation in CAD construction.

### 3 Improved lifting over algebraic points

Let  $c = \{\alpha\}$  be a 1-point cell in the induced CAD of 1-space that is a zero of some 1-level projection factor  $p(x)$ . In projecting the initial set  $A$ ,  $p(x)$  may have come about in a variety of ways: it may be a factor of a leading coefficient of an element of  $A$ , or a factor of a discriminant of an element of  $A$ , or a factor of the resultant of two elements of  $A$ . In fact, it may have several such derivations. Lifting is most expensive when  $p(x)$  is of high degree and has large coefficients, which generally means  $p(x)$  is a multiplicity one factor of a discriminant or resultant. After all, if  $p(x)$  is a multiplicity two factor, it is already half the generic expected degree of the resultant or discriminant. In fact, resultants and discriminants are generically irreducible, so that we expect (in one possibly very flawed sense of the word “expect”) to only deal with multiplicity one factors. This section will show that lifting over simple roots of discriminants and resultants can typically be done quite quickly and without having to resort to exact substitution and squarefree basis computation.

#### 3.1 Lifting over simple roots of resultants

In this section we consider the case in which the only derivation of  $p$  is as a multiplicity one factor of the resultant of two elements of  $A$ , call them  $P$  and  $Q$ . We will assume that we have already lifted over the sample points of  $c$ 's neighbors, which we'll call  $b$  and  $d$ , which is easily accomplished because both cells have rational sample points. Because none of the discriminants or leading coefficients of elements of  $A$  vanish anywhere in  $b \cup c \cup d$ , the roots of each individual element of  $A$  are non-intersecting, continuous functions over  $b \cup c \cup d$ . Because none of the resultants of pairs of elements of  $A$  other than  $P$  and  $Q$  vanish anywhere in  $b \cup c \cup d$ , the only possible intersections between sections of different elements of  $A$  would be between sections of  $P$  and  $Q$ , and those intersections would have to occur over  $c$ . Finally, Theorem 1 shows that because  $p(x)$  is a multiplicity one factor of  $res_y(P, Q)$ , there is exactly one intersection between sections of  $P$  and  $Q$ , and it must be simple - i.e. a true intersection and not a point of tangency.

By examining the roots of the elements of  $A$  over  $b$ 's sample point and  $d$ 's sample point we can determine which sections of  $P$  and  $Q$  intersect, because the  $k$ th section of  $P$  will be below the  $j$ th section of  $Q$  in one stack and vice versa in the other stack, as Figure 2 illustrates. Thus, we can isolate the roots of each element of  $A$  over  $x = \alpha$  using, for example, the interval Descartes method, and use the order of cells in the neighboring stacks to assign these isolating intervals to cells and determine which two roots of  $P$  and  $Q$  are actually the same.

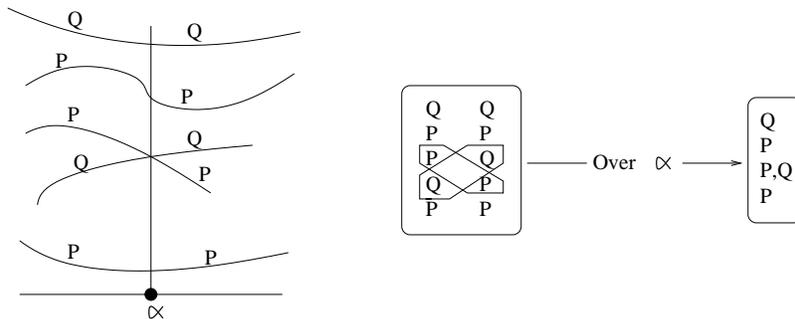


Figure 2: The order of sections of  $P$  and  $Q$  immediately left and right of point  $\alpha$  tell us exactly how the roots are arranged over  $\alpha$ .

### 3.2 Lifting over simple roots of discriminants

In this section we consider the case in which the only derivation of  $p$  is as a multiplicity-one factor of the discriminant of an element of  $A$ , call it  $Q$ . We will assume that we have already lifted over the sample points of  $c$ 's neighbors, which we'll call  $b$  and  $d$ , which is easily accomplished because both cells have rational sample points. Since none of the discriminants or leading coefficients of elements of  $A$  other than  $Q$  vanish anywhere in  $b \cup c \cup d$ , the roots of each individual element of  $A - \{Q\}$  are non-intersecting, continuous functions over  $b \cup c \cup d$ . Because none of the resultants of pairs of elements of  $A$  vanish anywhere in  $b \cup c \cup d$ , there are no intersections between sections of different elements of  $A$  over  $b \cup c \cup d$ . Finally, Theorem 2 shows that because  $p(x)$  is a multiplicity-one factor of  $disc_y(Q)$ , there are no singularities of  $Q$  over  $\alpha$ , and there is exactly one vertical tangent to  $Q$  over  $\alpha$ , and it is simple.

From the arrangement of the sections of the elements of  $A$  over  $b$  and  $d$ , we can determine the arrangements of the roots of the elements of  $A$  over  $\alpha$ , except that we may not be able to determine which two sections of  $Q$  come together to form a double root at  $x = \alpha$ , as Figure 3 illustrates. Using the interval Descartes method we can straightforwardly isolate the roots of all elements of  $A - \{Q\}$  over  $\alpha$ , and we already know how these roots are arranged with respect to one another. All that remains is to isolate the roots of  $Q$  over  $\alpha$  and determine which root is the double root. One could imagine a variety of methods for doing this without resorting to exact substitution and squarefree basis computation. I believe that the fastest and most straightforward way to do this is to simply use the interval Descartes method. It will attempt to isolate the roots of  $Q$  and it will fail, because  $Q$  is not squarefree over  $\alpha$ . The method starts with an interval in which it can determine *a priori* all roots of  $Q(\alpha, y)$  must lie, and it repeatedly divides the interval into smaller subintervals: those that it determines do not contain roots are thrown away, those that it determines contain exactly one

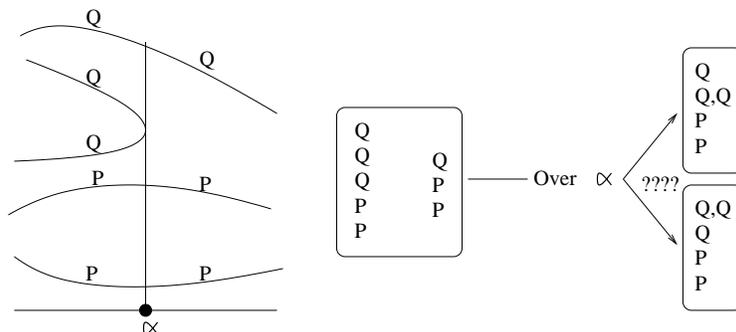


Figure 3: The order of sections of  $P$  and  $Q$  immediately left and right of point  $\alpha$  tell us how the roots are arranged over  $\alpha$ , except for the location of the double root of  $Q$ .

root are appended to the list of isolating intervals, and those that remain are further subdivided. This determination of the number of roots in an interval is based on counting the number of sign variations in the coefficients of certain polynomials. If the number of variations is two or more, the interval must be subdivided further. However, since the interval Descartes method represents each coefficient of  $Q(\alpha, y)$  as an interval containing the actual coefficient, it may happen that a coefficient's sign cannot be determined because the interval representing it contains zero. Thus, the number of coefficient sign variations cannot be determined, and the method must report that it failed to determine the number of roots in that interval. When the interval Descartes method fails to isolate the roots of  $Q(\alpha, y)$  and there is only one interval in which it failed, that interval must contain the double root of  $Q(\alpha, y)$ . Thus, even though the interval Descartes method "fails", our *a priori* knowledge about the roots of  $Q(\alpha, y)$  allows us to recover isolating intervals for all the roots and determine which interval contains the double root.

### 3.3 Lifting over roots of a leading coefficient

In this section we consider the case in which the only derivation of  $p$  is as a factor of a leading coefficient of an element of  $A$ , call it  $Q$ . Because the sizes of factors of leading coefficients are usually dwarfed by the sizes of factors of discriminants and resultants, this is not typically very interesting. However, we consider this case for completeness. We will assume that we have already lifted over the sample points of  $c$ 's neighbors, which we'll call  $b$  and  $d$ , which is easily accomplished because both cells are sectors, and thus have rational sample points. Because none of the discriminants or leading coefficients of elements of  $A$  other than  $Q$  vanish anywhere in  $b \cup c \cup d$ , the roots of each individual element

of  $A - \{Q\}$  are non-intersecting, continuous functions over  $b \cup c \cup d$ . Because none of the resultants of pairs of elements of  $A$  vanish anywhere in  $b \cup c \cup d$ , there are no intersections between sections of different elements of  $A$ . Thus, we know how all of the roots of the elements of  $A$  at  $x = \alpha$  are arranged, except that we don't know which sections of  $Q$  go off to vertical asymptotes over  $x = \alpha$ . However, we can isolate the roots of the reductum of  $Q$ , i.e.  $Q$  minus its leading term, and thereby get the roots of  $Q(\alpha, y)$ . We do not, in fact, need to worry about whether or not the leading coefficient of the reductum of  $Q$  vanishes at  $x = \alpha$ , since this would imply that  $p(x)$  also has a derivation as a factor of the discriminant of  $Q$ .

## 4 The Implementation

The alternate methods for lifting over simple roots of resultants and discriminants described in the previous section have been implemented. In this section we describe our implementation, while Section 5 reports on some experimental results it provided.

This implementation is based on QEPCAD, an implementation of quantifier elimination by partial CAD due to Hoon Hong, with subsequent improvements by many others. QEPCAD itself is built on top of the SACLIB library of C functions for computer algebra, which has been developed over the years by George Collins and many of his students and colleagues. One set of functions, due to Collins, Werner Krandick and Jeremy Johnson, is of particular interest for this paper, as it implements the interval Descartes method for root isolation [JK97]. There are SACLIB functions implementing the method with hardware doubles and also with arbitrary precision software floating-point numbers. Our implementation uses both — first attempting calculations with hardware floats and, the attempt failing, moving on to higher precision software floats. The current version does not do anything sophisticated with precision control when software floats are used. In fact, it simply performs all computations with 232-bit mantissa floating-point arithmetic, which will be more precision than necessary in many cases, and will be too little precision for some very large cases. Intelligent precision control will be part of a future investigation into improved use of floating-point computations in this context.

Essentially, we implemented the simplest possible versions of the improvements described in Section 3.1 and Section 3.2. When lifting over section cell  $c = \{\alpha\}$ , the improved method is used in the following two cases:

- $\alpha$  is a simple root of the resultant of a pair of elements of  $A$ , and is a root of no other resultant, discriminant or leading coefficient of an element of  $A$ . In this case, we implement the idea from Section 3.1, using the interval

Descartes method from SACLIB instead of QEPCAD’s usual lifting.

- $\alpha$  is a simple root of the discriminant of  $Q \in A$ , and is a root of no other discriminant or leading coefficient or resultant of pairs of elements of  $A$ . In this case we implement the idea from Section 3.2 using a slightly modified version of the interval Descartes method from SACLIB instead of QEPCAD’s usual lifting. The modified version simply keeps track of any intervals in which the method “fails”, which is information that we need, as described in Section 3.2.

Our implementation is really just a proof of concept and is lacking in many respects — though as shown in Section 5 it is none the less a substantial improvement over QEPCAD’s current lifting for 2D CAD’s. Our precision management, as already stated, is naive and limited — we simply use hardware doubles for our first attempt, if that fails we use 232-bit software floating-point arithmetic and, that failing, we fall back on exact techniques. Another limitation of the implementation is our response to failure of the interval arithmetic techniques, which is to fall back on the existing method for the entirety of that stack construction. Since the failure of interval root isolation for one projection factor does not require us to fall back on the existing method of root isolation for the other projection factors, we need not perform all of the failed stack construction with the old method. Finally, our implementation falls back on the old lifting method for any single-point cell  $\alpha$  that is a simple root of more than one resultant or discriminant. The improved lifting may still be used in these cases, it’s simply more complicated to implement. A more robust implementation addressing these problems should probably wait until other more substantial issues are explored, such as extending the new method to deal with higher multiplicity zeros of resultants and discriminants, and better adapting interval root isolation to this particular application.

## 5 Experiments

Of course we would like to test our method for 2-dimensional CAD construction against the usual method for a large representative set of 2-dimensional CAD construction problems. Unfortunately, there is no clear idea what a representative set looks like. We report on four types of CAD construction problems that hopefully provide a wide enough range of input types. All tests were run on a 360 MHz SUN Ultra-60. Note that the lifting method against which we test our implementation is that reported in [CJK], where real root isolation for the squarefree basis is done with the interval Descartes method.

## 5.1 Random Bivariate Polynomials

In this section we report on the performance of our implementation of the improved lifting method for 2D CAD's on sets of randomly generated bivariate polynomials. Random polynomials are a terrible case for the usual lifting method, because discriminants and resultants of random polynomials are irreducible, with the result that exact substitution and squarefree basis computations take place over very large algebraic extension fields. For the improved method, however, this is a very good case, because the points we lift over are always simple roots of resultants or discriminants. Random bivariate polynomials probably do not represent “typical” input, but they do provide an obvious place to start testing our new lifting method. Moreover, polynomials with coefficients that are only known approximately actually behave like random polynomials, in the sense that their discriminants and resultants are likely to be irreducible, so these experiments may tell us something about certain classes of “typical” polynomials.

Our test compares the time required for the new and old lifting methods to construct CAD's for pairs of randomly generated bivariate polynomials. Input pairs with different target total degrees, term densities, and coefficient sizes are tested. For each given input “size”, a single case is randomly generated. It would, of course, be better to average over many test cases of a given size, but the time required for these computations makes this infeasible. The old lifting method fails for many of these examples, usually after thousands of seconds, due to the “prime list” being exhausted. Various modular algorithms (in this case the algebraic polynomial gcd algorithm due to Encarnación [Enc95]) in the SACLIB system choose primes for modular algorithms from a list of prime numbers kept by the system. If a given calculation uses all of those primes and requires yet more, SACLIB exits with a FAIL message.

As the data from Figure 4 shows, the usual lifting method does not do well with random input while the new method is quite fast. To provide some kind of idea of the extent to which this makes new problems feasible, consider the following example. We generated two random, dense polynomials of total degree 20 with 10 bit coefficients, and were able to construct a CAD from the two polynomials in 196 seconds, 144 of which were spent in the projection phase! This problem is far beyond the scope of what is solvable using the old lifting method.

## 5.2 Polynomials defining “interesting” curves

In opposition to random polynomials, we consider a collection of decidedly “unrandom” polynomials — polynomials we have gathered that were deemed interesting because of the curves they define. As Figure 5 shows, these polynomials define curves that are singular and exhibit symmetry — both properties that

20% Term Density				
	10 Bit Coefficients		50 Bit Coefficients	
total degree	total time (sec)	lifting time (sec)	total time (sec)	lifting time (sec)
6	0.3 vs. 15.0	0.1 vs. 14.8	0.2 vs. 82.4	0.1 vs. 82.3
8	0.6 vs. 67.1	0.2 vs. 66.7	2.6 vs. fail	0.3 vs. fail
10	4.1 vs. 6176.5	0.8 vs. 6173.2	9.2 vs. fail	1.1 vs. fail
12	7.2 vs. fail	1.4 vs. fail	23.4 vs. fail	1.8 vs. fail

100% Term Density				
	10 Bit Coefficients		50 Bit Coefficients	
total degree	total time (sec)	lifting time (sec)	total time (sec)	lifting time (sec)
6	0.3 vs. 66.1	0.1 vs. 65.9	0.8 vs. fail	0.2 vs. fail
8	1.2 vs. 1762.9	0.3 vs. 1762.0	3.1 vs. fail	0.5 vs. fail
10	3.9 vs. fail	0.6 vs. fail	11.9 vs. fail	2.7 vs. fail
12	9.4 vs. fail	1.8 vs. fail	32.5 vs. fail	13.2 vs. fail

Figure 4: Timing data comparing the New vs. Old lifting method for pairs of randomly generated polynomials.

	Curve 1	Curve 2	Curve 3	Curve 4	Curve 5	Curve 6
						
tdeg	8	6	6	10	4	4
coef	6	7	6	8	17	5

Figure 5: A collection of “interesting” curves. this figure gives a picture, along with the total degree and coefficient bit length of the defining polynomial for the each curve.

render our lifting improvements often inapplicable. Thus, they are certainly not a good case for our method. On the other hand, they are a fairly good case overall for CAD construction! Our trials involve constructing CAD's for pairs of these curves, and constructing a CAD for all six curves together. Timing data for these experiments appear in Figure 6.

Curves 1 & 2	Time	Old	New	$\times$ speedup	
	Total	3.0	0.5	6.0	
	Lifting	2.8	0.3	9.3	
Curves 3 & 4	Time	Old	New	$\times$ speedup	
	Total	0.5	0.5	1.0	
	Lifting	0.1	0.1	1.0	
Curves 5 & 6	Time	Old	New	$\times$ speedup	
	Total	0.30	0.06	5.0	
	Lifting	0.27	0.03	9.0	
Curves 1 — 6	Time	Old	New	$\times$ speedup	
	Total	25.2	8.0	3.15	
	Lifting	23.6	6.4	3.69	

Figure 6: Timing data for CAD's involving “interesting” curves.

### 5.3 Resultants of pairs of random trivariate polynomials

The “interesting curves” of the previous section provided bad examples for the improved lifting method because of the presence of higher multiplicity factors in the discriminants of input polynomials. However, finding large polynomials defining “interesting” curves is somewhat difficult. So we consider constructing CAD's for resultants of pairs of random trivariate polynomials. Theorem 3.4 of [McC99] states that the discriminant of such a polynomial is not squarefree. Thus we know that there will be stack constructions for which the the new lifting method does not apply. None the less, the improved lifting produces substantial speedups and extends the scope of problems for which CAD's can be constructed. Figure 5.3 shows timings of experimental trials in which two random trivariate polynomials,  $p$  and  $q$ , were generated and their resultant,  $r$ , used as the sole input polynomial for CAD construction. It shows the new lifting method substantially outperforming the old, though not as decisively as with random bivariate polynomials. This is, of course, because exact lifting must be performed over the roots of the multiple factors of the discriminant of  $r$ . Were the ideas of this paper to be extended to deal with double roots of resultants and discriminants, we could expect much more substantial speedups, and we could tackle much larger problems. To give some idea of the size of the polynomials

tdeg $p, q$		Trial 1	Trial 2	Trial 3	Trial 4	Trial 5
$tdeg_p = 2$	New	0.02/0	0.03/0.01	0.05/0.01	0.09/0.05	0.04/0.01
$tdeg_q = 2$	Old	0.18/0.14	0.15/0.13	0.21/0.18	0.23/0.19	0.16/0.13
$tdeg_p = 3$	New	0.23/0.03	0.28/0.04	0.23/0.01	0.41/0.17	0.35/0.16
$tdeg_q = 2$	Old	9.6/9.4	8.8/8.6	7.5/7.3	10.7/10.4	6.9/6.7
$tdeg_p = 3$	New	23.8/21.8	23.3/21.3	2.1/0.07	20.8/19.0	19.3/17.2
$tdeg_q = 3$	Old	496/494	467/465	476/474	430/429	367/365
$tdeg_p = 4$	New	810/800	719/709	10.4/0.1	922/912	983/973
$tdeg_q = 3$	Old	> 2036	> 1949	> 1241	> 2148	> 2210

Figure 7: Times (in seconds) for trials of New vs. Old lifting method. Times given are “total time” / “lifting time”. Input is  $res_z(p, q)$ , where  $p$  and  $q$  are two random polynomials in  $x, y$ , and  $z$ , with 8-bit coefficients, 50% term density, and varying total degrees. The Old lifting method failed due to exhaustion of the prime list for all inputs from the last row, so the numbers given are the time taken up to the point of failure.

considered here, the resultant  $r$  from Trial 5 of the data set with  $tdeg_p = 4$  and  $tdeg_q = 3$  has degree 12 in  $y$  and in  $x$ , has coefficients from 40 to 60 bits in length, and consists of 91 terms.

#### 5.4 Discriminants and Resultants of pairs of random trivariate polynomials

We consider sets of polynomials consisting of the two discriminants and the pairwise resultant of two randomly generated trivariate polynomials. From Theorem 3.4 of [McC99] we see that the discriminants of the resulting three bivariate polynomials will not be squarefree, and from Theorem 3.3 of [McC99] we see that at least two of the three resultants will also not be squarefree. Thus, input polynomials generated in this way provide an especially bad case for the improved lifting. Figure 8 shows timing data for a number of experimental trials. For given total degrees  $d_1$  and  $d_2$ , and term density  $D$ , we generated two random polynomials in  $x, y$ , and  $z$  with total degrees  $d_1$  and  $d_2$ , term density  $D$ , and 8-bit coefficients. The bivariate input polynomials were  $\{disc_z(p), disc_z(q), res_z(p, q)\}$ .

## 6 Possible applications and future research

This paper presents an improved lifting method for 2D CAD construction. Experimental evidence suggests that the improved method not only substantially

Term Density 20%				
$tdeg_p$	2	3	3	4
$tdeg_q$	2	2	3	3
New	0.15/0.11	0.77/0.21	2.0/0.4	28/1
Old	0.22/0.18	9.8/9.2	10.6/9.0	> 1352

Term Density 100%				
$tdeg_p$	2	3	3	4
$tdeg_q$	2	2	3	3
New	0.13/0.07	3.1/2.2	24/20	1596/1561
Old	0.31/0.26	19.7/18.8	568/563	> 1268

Figure 8: Timing data for New vs. Old lifting method. Times are given in seconds, “total time”/“lifting time”. Random trivariate polynomials  $p$  and  $q$  are generated with given total degrees and term densities, and with 8-bit coefficients. The input polynomials for 2D CAD construction consist of  $disc_z(p)$ ,  $disc_z(q)$ , and  $res_z(p, q)$ . An entry “>  $n$ ” indicates that lifting failed due to prime list exhaustion after  $n$  seconds.

speeds up the overall 2D CAD construction process, but also brings problems that were completely inaccessible using the old lifting method well within the realm of what can be computed in a reasonable amount of time. This section will briefly discuss potential further improvements to the method, the possible extension of the method to 3D CAD construction, and one potential application of CAD in light of these improvements.

## 6.1 Improvements of the method

There are, as has been noted, a number of ways that our implementation of the new lifting method could be improved. But there are also some substantial areas for improvement in the method itself.

First of all, it may be possible to optimize the interval Descartes method for this problem. For example, as presented in [JK97], the method may fail for all precisions even if the input interval polynomial contains only squarefree polynomials. It should be possible to ensure that the method works at some precision for any input interval polynomial containing only squarefree polynomials. Optimizing the interval Descartes method for non-squarefree polynomials also bears investigation, as does refinement of intervals which may contain double roots.

Secondly, it would be nice to extend the method described in this paper to lifting over multiple roots of discriminants and resultants. For example, if we lift over some  $\alpha$  that is a double root of the discriminant of polynomial  $p$ , we may end up

with exactly two “fail” intervals. We would need to determine how many roots were in each of the intervals, and what their multiplicities were. Let  $(i_1, i_2)$  be the isolating interval for  $\alpha$ , and let  $(j_1, j_2)$  and  $(k_1, k_2)$  be the two fail intervals. In effect reversing the “box method” of [ACM84b], we construct the “boxes”  $B_1 = [i_1, i_2] \times [j_1, j_2]$  and  $B_2 = [i_1, i_2] \times [k_1, k_2]$ , and determine the intersection points of  $p(x, y)$  with  $B_1$  and  $B_2$ . For some configurations of intersections we may be able to determine the arrangement of the roots of  $p(\alpha, y)$ , as is indicated in Figure 9.

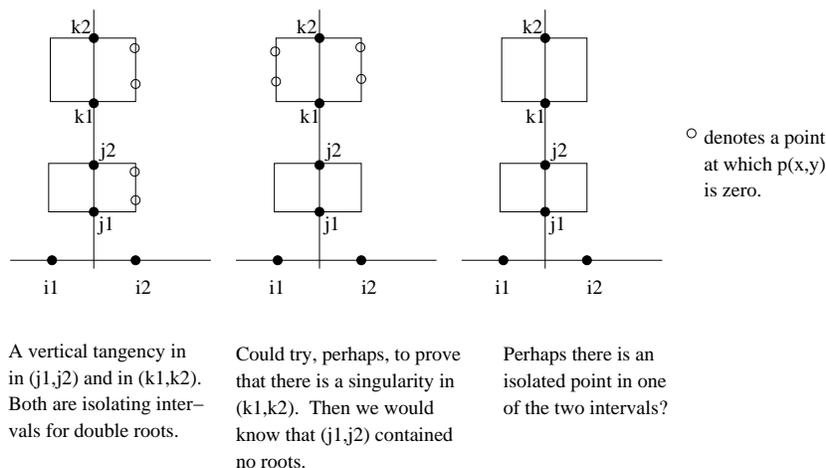


Figure 9: Using “boxes” to determine what’s happening in a “fail interval”.

## 6.2 Extensions to 3D CAD construction

In the construction of cylindrical algebraic decompositions of  $\mathbb{R}^3$ , the bottleneck is in lifting over single-point cells in an induced CAD of  $\mathbb{R}^2$ . These points are of the form  $(\alpha, \beta)$ , where  $\alpha$  is the root of some polynomial  $M(x)$ , and  $\beta$  is a root of some polynomial  $A(\alpha, y)$ . Lifting means isolating and ordering the roots of a set  $S$  of polynomials in  $\mathbb{Q}(\alpha, \beta)[z]$ , namely the set of polynomials resulting from substituting  $x = \alpha$  and  $y = \beta$  into each element of  $P_3$ . The lifting algorithm does this by:

1. computing a polynomial  $N(x)$  and a root  $\gamma$ , such that  $\mathbb{Q}(\gamma)$  contains  $\alpha$  and  $\beta$ ,
2. computing  $B$ , a squarefree basis for  $S$ , where  $B \subset \mathbb{Q}(\gamma)[z]$ , and
3. isolating and ordering the real roots of the elements of  $B$ .

The first step, computing the primitive element  $\gamma$ , is quite expensive, and  $N$ , the minimal polynomial for  $\gamma$ , is of high degree. Thus, the second step is quite expensive as well. Using the interval Descartes method, the final step can be done fairly quickly.

Were we to develop an 3D analogue of the method described in this paper, we could lift over some of these points (depending on multiplicities) without steps one or two. For each trivariate projection factor  $f(x, y, z)$ , we would construct an interval polynomial containing  $f(\alpha, \beta, z)$  and use the interval Descartes method to isolate the roots of this interval polynomial. If this succeeds, we will have isolated the roots of  $f(\alpha, \beta, z)$  without computing primitive elements. We could then order the roots of different projection factors using information about the order of roots over neighboring cells in the induced CAD. There are many issues to work out with such a scheme, but the potential time savings are quite large, because computing the primitive element  $\gamma$  is so expensive, and computing in the large extension defined by  $\gamma$  is so expensive.

### 6.3 Potential application

The improvements described in this paper dramatically speed up CAD construction for certain types of problems — most dramatically for sets of non-singular polynomials. Perhaps CADs may therefore be applied to problems for which they were previously deemed to be too expensive to compute.

For example, Keyser et al. [KCMK00] report on algorithms and software for performing exact manipulation of algebraic points and curves, with the intended field of application being Computer Aided Design (the other CAD). One problem they consider is computing arrangements of non-singular planar algebraic curves. The same information they compute may be computed by CAD construction augmented with adjacency computation. Because we don't have the data sets they report on in their paper, we cannot really compare the time required by the improved CAD construction and adjacency computation with the time required by their method. They give the number and coefficient bit length of polynomials in their trials, but not degree or term density. They report, for example, that an arrangement of 3 curves defined by polynomials with 25-bit coefficients was computed in 8.38 seconds on a 40 MHz Pentium II machine. The picture given of the 3 curves shows that their degrees in  $y$  must be at least 2, but not much more can be said. To do some rough comparison, we generated 3 random, dense polynomials of total degree 5 with 25-bit coefficients, and constructed a CAD with adjacency information for them. This took 0.7 seconds using the improved lifting, and 179 seconds using the original lifting method. Whether this comparison of CAD versus the method of Keyser et al. is meaningful depends, of course, on the degrees and term densities used in their experiment, and whether or not the polynomials used there were random

(although we do know they were non-singular). However, it does provide some evidence that with the old lifting method CAD does not provide a competitive alternative to the method described in Keyser et al., whereas with the new lifting method it does.

## 7 Acknowledgments

I would like to thank Werner Krandick for his help in understanding the interval Descartes method, Josef Schicho and Will Traves for answering my questions about algebraic geometry, and George Nakos for helping me with my proofs. This work was supported by a grant from the Naval Academy Research Council.

## 8 Appendix

This section contains theorem statements and proofs for results used in this paper. Note that for polynomials  $P, Q \in \mathbb{R}[x, y]$  with resultant  $r(x) = \text{res}_y(P, Q)$ , we have  $\text{res}_y(P(x - a, y - b), Q(x - a, y - b)) = r(x - a)$ . When proving many statements about polynomials and their resultants and discriminants at some arbitrary point, this allows us to assume that the point is in fact the origin.

**Theorem 1** *Let  $P(x, y)$  and  $Q(x, y)$  be relatively prime, squarefree polynomials with real coefficients, and let their degrees in  $y$  be  $m$  and  $n$  respectively. Let  $P_m(x)$  and  $Q_n(x)$  be their leading coefficients, and let  $r(x) = \text{res}_y(P, Q)$ . Let  $\alpha \in \mathbb{R}$  be a simple root of  $r$  that is not a root of  $\text{disc}_y(P)$ ,  $\text{disc}_y(Q)$ ,  $P_m$ , or  $Q_n$ . The real curves defined by  $P$  and  $Q$  have exactly one common point with  $x$ -coordinate  $\alpha$ , and they are not tangent at this point.*

PROOF. Since  $\alpha$  is not a root  $\text{disc}_y(P)$  or  $\text{disc}_y(Q)$ , the curves defined by  $P$  and  $Q$  do not have vertical tangents over  $\alpha$ . By Theorem 3, there can be no non-vertical points of tangency over  $\alpha$  between the curves defined by  $P$  and  $Q$  either. Thus, any common points between the curves defined by  $P$  and  $Q$  must be at simple intersections. Since  $\alpha$  is a root of  $r$ , but not of either  $P_m$  or  $Q_m$ , there is at least one common root of  $P$  and  $Q$  over  $x = \alpha$ . Theorem 5 shows that there is in fact exactly one common root, and since  $P(\alpha, y)$  and  $Q(\alpha, y)$  are both real polynomials, this root must be real.  $\square$

**Theorem 2** *Let  $P(x, y) = P_m y^m + \dots + P_1 y + P_0$  be a squarefree polynomial with real coefficients that is primitive as a polynomial in  $y$ . If  $\alpha \in \mathbb{R}$  is a simple*

root of  $\text{disc}_y(P)$  and  $P_m(\alpha) \neq 0$ , the curve defined by  $P$  has no singularities and exactly one vertical tangent over  $x = \alpha$ . Moreover, the vertical tangent is simple and at a point with real coordinates.

PROOF. Theorem 4 shows that  $P$  has no singularities over  $\alpha$ . A similar argument shows that  $P$  has no points of vertical tangency at which  $\partial^2 P / \partial y^2$  also vanishes, so any tangencies are simple. What remains to be proven is that there is exactly one point of vertical tangency, and that its  $y$ -coordinate is real. Since  $\text{disc}_y(P) = 1/P_m \text{res}_y(P, P')$ ,  $\alpha$  is a simple root of  $\text{res}_y(P, P')$ . Therefore, Theorem 5 shows that there is exactly one common root of  $P$  and  $P'$  over  $x = \alpha$ . Since  $P$  and  $P'$  are real polynomials, this root must be real.  $\square$

**Theorem 3** *Let  $P, Q \in \mathbb{R}[x, y]$  be relatively prime polynomials. Let  $\alpha$  be a point at which the discriminants and leading coefficients of  $P$  and  $Q$  as polynomials in  $y$  are all nonzero. If the curves defined by  $P$  and  $Q$  have a non-vertical mutual tangency at some point  $(\alpha, \beta)$ , then  $\alpha$  is a multiple root of  $\text{res}_y(P, Q)$ .*

PROOF. Without loss of generality assume  $(\alpha, \beta) = (0, 0)$ . Let  $y = kx$  be the mutual tangent line, then

$$P = a(y - kx) + \text{terms of total degree} \geq 2$$

Since the discriminant of  $Q$  does not vanish at  $x = 0$ , the roots of  $Q$  may be defined as analytic functions of  $x$  over some neighborhood of zero. Let  $\gamma_1, \dots, \gamma_n$  be the root functions for  $Q$ . Without loss of generality, suppose  $\gamma_1$  defines the portion of the curve passing through  $(0, 0)$ . By a well known identity,  $r(x) = (-1)^{mn} Q_n^m \prod_{i=1}^n P(x, \gamma_i(x))$ . Thus,

$$\begin{aligned} r'(x) = & \frac{d}{dx}[P(x, \gamma_1(x))] \cdot [(-1)^{mn} Q_n^m \prod_{i=2}^n P(x, \gamma_i(x))] \\ & + \\ & P(x, \gamma_1(x)) \cdot \frac{d}{dx}[(-1)^{mn} Q_n^m \prod_{i=2}^n P(x, \gamma_i(x))] \end{aligned}$$

The second term vanishes at  $x = 0$ , since  $P(0, \gamma_1(0)) = P(\alpha, \beta) = 0$ . If we can prove that  $d/dx[P(x, \gamma_1(x))]$  is zero at  $x = 0$  the first term would also clearly vanish, and having shown that  $r'(0) = 0$ , we would be done.

$$\begin{aligned} dP(x, \gamma_1(x))/dy &= P_x(x, \gamma_1(x)) + P_y(x, \gamma_1(x)) \cdot \gamma_1'(x) \\ &= -ak + \text{non-constant terms} + (a + \text{non-constant terms})\gamma_1'(x) \end{aligned}$$

Evaluating this at  $(0, 0)$  and keeping in mind that  $\gamma_1(0) = k$ , we get

$$-ak + 0 + (a + 0)k = 0$$

Since  $r(0) = r'(0) = 0$ ,  $\text{res}_y(P, Q)$  has a multiple root at  $x = 0$ .  $\square$

**Theorem 4** Let  $P(x, y) = P_m y^m + \dots + P_1 y + P_0$  be a squarefree polynomial with real coefficients that is primitive as a polynomial in  $y$ . If  $P$  has a singularity at  $(\alpha, \beta) \in \mathbb{R}^2$ , then  $\alpha$  is a multiple root of  $\text{disc}_y(P)$ .

PROOF. Without loss of generality, assume that  $(\alpha, \beta) = (0, 0)$ .  $P$  is singular at  $(0, 0)$ , so  $P$ ,  $\partial P/\partial y$  and  $\partial P/\partial x$  all vanish at  $(0, 0)$ . Thus, for some  $A, B \in \mathbb{R}[x]$  and  $b, c \in \mathbb{R}$ , we get  $P_0 = x^2 A$ ,  $P_1 = x^2 B + bx$ , and

$$\text{disc}_y(P) = 1/P_n \begin{vmatrix} P_n & \cdots & 0 & 0 \\ & \ddots & \vdots & \vdots \\ 0 & \cdots & x^2 A & 0 \\ 0 & \cdots & bx + x^2 B & x^2 A \\ nP_n & \cdots & 0 & 0 \\ & \ddots & \vdots & \vdots \\ 0 & \cdots & bx + x^2 B & 0 \\ 0 & \cdots & c + xC & bx + x^2 B \end{vmatrix}$$

If  $b = 0$ ,  $x^2$  may be factored from the last column, and the theorem is proved. If  $x \neq 0$ ,  $x$  may be factored from the last column and, after an elementary column operation,  $x$  may also be factored from the second to last column.  $\square$

**Theorem 5** Let  $P, Q \in \mathbb{C}[x, y]$  be primitive in  $y$  and let  $R(x) = \text{res}_y(P, Q)$  be non-zero. If  $\alpha$  is a simple root of  $R$  and at least one of the leading coefficients is non-zero at  $\alpha$ , there is exactly one common root of  $P(\alpha, y)$  and  $Q(\alpha, y)$ .

PROOF. Without loss of generality, assume  $\alpha = 0$ . Suppose there are two common roots,  $\beta_1$  and  $\beta_2$ . If  $\beta_1 = \beta_2$ , i.e. if the curves defined by  $P$  and  $Q$  are tangent at  $(\alpha, \beta_1)$ , we see by Theorem 3 that  $\alpha$  must be a double root of  $R$ , which contradicts our hypothesis. Thus,  $\beta_1 \neq \beta_2$ . Without loss of generality, assume  $\beta_1 = 0$ .

Let  $R_\epsilon(x) = \text{res}_y(P(x, y(1 - \epsilon)), Q(x, y))$ . As  $\epsilon$  approaches zero,  $R_\epsilon(x)$  approaches  $R(x)$ . Moreover, for small non-zero  $\epsilon$  the origin is the only common zero of  $P(x, y(1 - \epsilon))$  and  $Q(x, y)$  over  $x = 0$ . However, in some small neighborhood of  $(0, \beta_2)$  there is another common root of  $P(x, y(1 - \epsilon))$  and  $Q(x, y)$ . To see this, change coordinates by rotating both curves so that all distinct roots have distinct  $x$ -coordinates and both transformed polynomials are still primitive in  $y$ . Let  $P^*$  and  $Q^*$  be the rotated polynomials and  $(\alpha^*, \beta^*)$  be the rotated  $(0, \beta_1)$ . Any sufficiently small perturbation of  $P$  produces a small perturbation of  $P^*$ , which produces a small perturbation of  $\text{res}_y(P^*, Q^*)$ . Thus, there will be a root of  $\text{res}_y(P^*, Q^*)$  near in a small neighborhood of  $\alpha^*$ , and a common zero of  $P^*$  and  $Q^*$  must exist over that root. A continuity argument shows then that this common zero must be near  $(\alpha^*, \beta^*)$ .

Let  $\alpha_\epsilon$  be the  $x$ -coordinate of the common root of  $P(x, y(1-\epsilon))$  and  $Q(x, y)$  near  $(0, \beta_2)$ ; clearly  $\alpha_\epsilon$  must be a root of  $R_\epsilon$ . As  $\epsilon$  approaches zero,  $\alpha_\epsilon$  approaches zero. Thus, as  $\epsilon$  approaches zero we have two distinct roots of  $R_\epsilon$ ,  $x = 0$  and  $x = \alpha_\epsilon$ , both approaching zero.

Now, the coefficients of  $R_\epsilon$  are symmetric functions of the roots of  $R_\epsilon$  — two of which are  $\alpha_\epsilon$  and 0. Let  $n$  be the degree of  $R_\epsilon$ , and let  $c$  be the leading coefficient of  $R_\epsilon$ . The constant coefficient is  $c$  times the product of all  $n$  roots of  $R_\epsilon$ , and is therefore zero. The coefficient of  $x$  is a linear combination of all products of  $n - 1$  roots. All such products are zero, or contain  $\alpha_\epsilon$ . Thus, the coefficient of  $x$  is divisible by  $\alpha_\epsilon$ . As  $\epsilon$  approaches zero, the coefficient of  $x$  in  $R_\epsilon$  approaches zero as well. Thus,  $x^2$  divides  $R(x)$ , which contradicts the hypothesis of our theorem. [Note: I think this proof can basically be extended by induction to say that there are exactly  $k$  common roots of  $P$  and  $Q$  (including multiplicities and roots at infinity) over an order  $k$  root of  $\text{res}_y(P, Q)$ . This could be useful in extending the lifting method described here. ]  $\square$

## References

- [ACM84a] D. S. Arnon, G. E. Collins, and S. McCallum. Cylindrical algebraic decomposition I: The basic algorithm. *SIAM Journal on Computing*, 13(4):865–877, 1984.
- [ACM84b] D. S. Arnon, G. E. Collins, and S. McCallum. Cylindrical algebraic decomposition II: An adjacency algorithm for the plane. *SIAM Journal on Computing*, 13(4):878–889, 1984.
- [Arn83] D. S. Arnon. Topologically reliable display of algebraic curves. In *Proceedings of SIGGRAPH*, pages 219–227, 1983.
- [Bro01] C. W. Brown. Simple CAD construction and its applications. *Journal of Symbolic Computation*, 31(5):521–547, May 2001.
- [CA76] G. E. Collins and A. Akritas. Polynomial real root isolation using Descartes’ rule of signs. In R. D. Jenks, editor, *Proc. of the 1976 ACM Symposium on symbolic and algebraic computation*, pages 272–275, 1976.
- [CH91] G. E. Collins and H. Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12(3):299–328, Sep 1991.
- [CJ98] B.F. Caviness and J. R. Johnson, editors. *Quantifier Elimination and Cylindrical Algebraic Decomposition*. Texts and Monographs in Symbolic Computation. Springer-Verlag, 1998.

- [CJK] G. E. Collins, J. R. Johnson, and W. Krandick. Interval arithmetic in cad computation. Submitted to *Journal of Symbolic Computation*, 2001.
- [Col75] G. E. Collins. Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In *Lecture Notes In Computer Science*, volume Vol. 33, pages 134–183. Springer-Verlag, Berlin, 1975. Reprinted in [CJ98].
- [DS96] A. Dolzmann and T. Sturm. Redlog - computer algebra meets computer logic. Technical Report MIP-9603, FMI, Universität Passau, 1996.
- [Enc95] M. J. Encarnación. Computing gcds of polynomials over algebraic number fields. *Journal of Symbolic Computation*, 20:299–313, 1995.
- [JK97] J. R. Johnson and W. Krandick. Polynomial real root isolation using approximate arithmetic. In *Proc. International Symposium on Symbolic and Algebraic Computation*, pages 225–232, 1997.
- [KCMK00] J. Keyser, T. Culver, D. Manocha, and S. Krishnan. Efficient and exact manipulation of algebraic points and curves. *Computer-Aided Design*, 32:649–662, 2000.
- [McC99] S. McCallum. Factors of iterated resultants and discriminants. *Journal of Symbolic Computation*, 27:367–385, 1999.