Root comparison techniques applied to computing the additively weighted Voronoi diagram

Menelaos I. Karavelas* Ioannis Z. Emiris†

Abstract

This work examines algebraic techniques for comparing quadratic algebraic numbers, thus yielding methods for deciding key predicates in various geometric constructions. Our motivation and main application concerns a dynamic algorithm for computing the additively weighted Voronoi diagram in the plane. We propose efficient, exact, and complete methods, which are crucial for a fast and robust implementation of these predicates and the overall algorithm. Our first contribution is to minimize, on the one hand, the algebraic degree of the computed quantities, thus optimizing precision and, on the other hand, the total number of arithmetic operations. We focus on the hardest predicate, which involves quadratic polynomials, and detail the corresponding algorithms, which are based on polynomial Sturm sequences; ancillary tools include geometric invariants, multivariate resultants, and polynomial factorization. Our last contribution is a general and efficient implementation, which has been extensively tested in order to demonstrate the practical performance of our methods and the improvements achieved over existing approaches.

Keywords: Computational Geometry, Symbolic Computation, Voronoi Diagrams, Algebraic Predicates.

1 Introduction

In this paper we study techniques for comparing quadratic algebraic numbers. This is an instance of the more general problem of deciding the order between the real roots of given polynomial equations. This turns out to be an important predicate in several geometric constructions, as illustrated below. Manipulating algebraic numbers is a vast problem in real algebraic geometry, and admits a variety of approaches, whose full description goes beyond our scope. Instead, we develop and compare methods suitable to the case of quadratic univariate polynomials. This problem is by itself important in building geometric software, as is manifest by the related efforts mentioned in the sequel.

Our main motivation comes from Voronoi diagrams, which are among the most studied constructions in computational geometry due to their numerous applications, including motion planning and collision detection, communication networks, graphics, and growth of microorganisms or plants. This paper considers the planar *Additively Weighted Voronoi diagram*, or simply AW-Voronoi diagram. The input consists of a set of points and a set of weights associated to them; these are the weighted points or *sites*. We denote the Euclidean distance as $d(\cdot,\cdot)$ and define the distance $\delta(p,B)$ between a point $p \in \mathbb{R}^2$ and a site B as $\delta(p,B) = d(p,b) - r$, where $b \in \mathbb{R}^2$ is the center (or point) of B and r its weight. The AW-Voronoi diagram is the subdivision of the plane induced by assigning each point p to the nearest site with respect to the distance function $\delta(p,\cdot)$. If all weights are positive, the AW-Voronoi diagram is the Voronoi diagram for a set of circles. In contrast to the usual Euclidean Voronoi diagram for points, sites in an AW-Voronoi diagram may have empty Voronoi cell; these sites are called trivial.

There have been several algorithms for this problem, e.g. [3, 10, 12, 15, 18, 23], however the question of completely evaluating the predicates has seldom been treated and even less often implemented. In particular, [12, 23] discuss the predicates required, but they are rather complicated. The algorithm presented in [18] treats Voronoi diagrams in an abstract way and thus requires the predicates as input. In [17], an implementation of the Delaunay triangulation of the input points is used, followed by edge flips, in order to arrive at the AW-Voronoi diagram. However, the algorithm has quadratic worst-time complexity, it is off-line and it handles neither intersecting nor trivial sites. More recently, [2] has examined one important predicate in the algorithm of [3], which is nonetheless of quadratic complexity and off-line. Interestingly, this predicate is decided by algebraic expressions of maximum degree 16 in the input parameters.

The algorithm of [15] is fully dynamic, i.e., sup-

*INRIA Sophia-Antipolis, 2004 route des Lucioles, BP 93, 06902 Sophia-Antipolis, France; email: Menelaos.Karavelas@sophia.inria.fr

†Department of Informatics & Telecoms, University of Athens, Greece and INRIA Sophia-Antipolis, France; email: emiris@di.uoa.gr

ports arbitrary insertions and deletions of sites. Its asymptotic complexity is quadratic in the worst case, in terms of the total number of input sites, but on the average it is $O(nT(h) + h \log h)$, where n, h are, respectively, the number of input sites and the number of non-trivial sites, and $T(h)$ bounds the complexity of locating the nearest neighbor of a given site among all non-trivial sites. Moreover, experimental evidence suggests an $O(n \log h)$ behavior.

This algorithm requires 5 main predicates, which are implemented by 9 primitive operations. A full description of these predicates and primitives is beyond the scope of this paper; see [14, 15]. Here, we focus on algorithms for evaluating one of these predicates, which is the hardest from the point of view of algorithm design and algebraic complexity. It essentially reduces to comparing two algebraic numbers defined by quadratic polynomials. Our algorithms have been used in an implementation of all the predicates required by the algorithm in [15]. The experimental results in Section 5 compare different approaches to implementing the predicates of the overall algorithm, and, in particular, the hardest one.

Since the studied predicate amounts to comparing two quadratic roots, our methods apply to general problems dealing with algebraic numbers of low degree, thus leading to several potential applications; see also Section 1.1. In particular, techniques for exact comparisons of algebraic numbers are at the heart of software packages in computational geometry and solid modeling, such as CORE, LEDA, and MAPC [11, 20, 16]. Our experiments examine different approaches and compare their practical behavior on the basic algebraic operations with, but also without, reference to the Voronoi diagram under construction.

Predicate evaluation must be efficient and exact, in order to be fast and robust in practice, as well as complete in order to cover all input cases, including degeneracies. The goal of efficiency refers first to minimizing the algebraic degree of the computed quantities in the input parameters, thus optimizing the precision required for exact arithmetic. This is becoming nowadays a question that heavily influences algorithm design in computational geometry, e.g. [5, 19, 6]. Besides algebraic degree, a second and related task is to minimize the total number of arithmetic operations. This twofold goal corresponds roughly to minimizing bit complexity, which is the most realistic measure reflecting the running time, when using exact rational arithmetic. Completeness refers to applying our methods and analysis to arbitrary inputs, including degeneracies. Our algorithms make use of polynomial Sturm sequences, the classical geometric invariants, multivariate resultants, Descartes' rule of sign, and multivariate polynomial factorization.

This paper is structured as follows. The next subsection discusses existing work for the specific problem, and states our main results. Section 2 formally defines the examined predicate in algebraic terms, and relates it to the geometric problem. Section 3 describes our algebraic methods. Section 4 applies our methods to ordering quadratic algebraic numbers, and Subsection 4.1 discusses degenerate cases. These two sections prove our main claim on the maximum algebraic degree of the tested quantities. The implementation and several experimental results are described in Section 5. We conclude with directions of further work.

1.1 Past and new results. We discuss existing approaches to comparing two specific roots of two quadratic polynomials, and indicate our contributions. In the sequel, when we refer to degrees, we refer to the algebraic degrees of the corresponding expressions with respect to the input data of the AW-Voronoi diagram algorithm, i.e., the coordinates of the center of the weighted point and its weight.

The most straightforward approach is to use radicals in order to express the specific roots that must be compared, then develop these expressions with the required squaring in order to arrive at an evaluation of rational quantities. For the predicate in question, the largest tested expression has degree 36 in the input parameters, which is more than twice that encountered in our Sturm-based method of Section 4. We may consider the corresponding computation tree, and assume that all branches are equally likely. At each node, we multiply the number of required arithmetic operations by the algebraic degree of the computed quantity, thus obtaining a total of 180 for ordering the larger roots of the two polynomials: It is substantially higher than the methods detailed in the sequel.

Certain existing generic software packages, such as CORE or LEDA [11, 20], rely on estimates on the bit precision required to decide the sign of an expression, defined by its evaluation tree. Their capabilities are of course very general. Our experiments consider the number type of LEDA reals and show that our implementation runs significantly faster on the specific predicate and the overall algorithm.

A first approach is letting $x_1 = t + x_2$, where t is a new variable whose sign indicates the order between the x_i roots. Comparing the roots of $f_1(t + x_2), f_2(x_2)$ by expressions of nested radicals reduces the algebraic degree to 28. Even better, the Sylvester resultant R of $f_1(t + x_2), f_2(x_2)$ with respect to x_2 is a polynomial in t. By studying its roots, we can solve the problem at

hand. This turns out to be equivalent to the resultant-based approach of [9] for deciding the main predicates in an algorithm computing arrangements of circular arcs defined by the intersection of a circle and a straight line. The primitive studied in [9] can be expressed precisely as an ordering question between two quadratic algebraic numbers. We review their resultant-based method and show how our techniques can yield an improvement in terms of algebraic degree as well as number of operations.

Polynomial $R(t)$ can be obtained as the resultant of $f_0(x_1, x_2) := -t + x_1 - x_2, f_1(x_1), f_2(x_2)$ with respect to x_1, x_2. This can be seen as an instance of the u-resultant, where the u-polynomial is f_0. For background on multivariate resultants, see Section 3. In [14] we elaborate their application to our problem, including the computation of $R(t)$ as a single matrix determinant. Let us write $R(t) = \sum_i R_i t^i$, where $R(t) = (\alpha_1\alpha_2)^2 t^4 + 4\alpha_1\alpha_2 J t^3 + (4J^2 + 2\alpha_1\alpha_2 K)t^2 + 4KJt + (G^2 - 4JJ')$, where α_i are the leading coefficients of f_i and J, J', K, G, Δ_i are functions of these coefficients defined in Section 4. By applying Descartes' rule of sign (cf. Section 3) to the signed coefficients of $R(t)$, it is possible to know the sign of the roots of t in almost all cases. See, for illustration, Table 3, where the cases are defined in Table 1. In certain cases, an additional test is required on the sign of $E := \Delta_1\alpha_2^2 - \Delta_2\alpha_1^2$. The maximum degree of any computed quantity in the input parameters is 20. For comparing the two larger roots, and assuming that each branch in an evaluation procedure is equally likely, the expected number of operations is $12\frac{3}{4}$, which becomes $150\frac{1}{4}$ when we multiply the count of each operation by the maximum degree of its arguments. If we suppose that all cases are equally likely, then we require $12\frac{3}{4}$ operations on the average.

In the above cost computations we assume that Δ_1 and Δ_2 are given. Such an assumption is realistic, given that the primitives that are of interest in this paper are at the bottom-most levels of the overall evaluation of the predicates, and the quantities Δ_1 and Δ_2 have already been computed at higher evaluation levels. Moreover, we would like to stress that the evaluated estimate of the bit complexity (average of number of operations multiplied by degree) is of qualitative nature. In particular, since we use the maximum degree of the quantities involved in the computation of a node, we over-estimate the actual cost of computing the intermediate quantities. On the other hand, we under-estimate the cost of multiplications, which are superlinear in the bit size of the multiplied quantities.

By definition, resultants offer the smallest condition for the solvability of a system of $n+1$ polynomials in n variables; in our case $n = 2$. This implies that the algebraic degree of 20 is optimal, provided that the input polynomials have *generic* coefficients. However, this is clearly not the case. By writing these coefficients in terms of the input parameters, we were able to factorize the constant coefficient in $R(t)$, which is the only one of degree 20; this is discussed in Section 4. The bottleneck now becomes the sign of E, a quantity of degree 18.

Our contribution is a general approach based on Sturm sequences. We show that the expected number of operations under the 3 measures used above is always smaller than the corresponding numbers when using only resultants. This improvement can be readily applied to the predicates in [9]. More importantly, our method can better exploit factorization to reduce the maximum algebraic degree appearing in the predicates for the AW-Voronoi diagram down to 16. Experimental results illustrate our improvements. Interestingly enough the quantities that are being tested by our Sturm sequence approach are the same with those obtained using other algebraic techniques [21], including quadrics' theory, application of a theorem by Hermite and Bezoutian matrices.

Further related work includes [1], in particular the χ_2 primitive, which offers an alternative way for deciding our predicate by reducing the decision to the sign of a determinant of radicals. But this approach involves expressions of degree up to 18 for generic input, whereas with ours the corresponding degree is 14. Moreover the average number of operations is more than 15 and the corresponding count, when the algebraic degree is incorporated, exceeds 174.

2 Order on bisector

The main predicate in the algorithm under consideration determines the type of the conflict region of a query site with a given Voronoi edge. Depending on whether this Voronoi edge lies on an infinite or finite bisector of two input sites, certain subpredicates will be called. The hardest case is when the bisector is finite, in which case the main subpredicate is OrderOnBisector. This decides the order of two points on the (oriented) bisector and uses, in its turn, two primitives: The Orientation primitive, which is straightforward and well-known in computational geometry, and the RadiDifference primitive, which is the focus of this paper. More precisely, the Orientation primitive is applied to two input points and the Voronoi vertex of three sites. It can be decided for arbitrary inputs, in a straightforward manner, using expressions of degree at most 14 [14].

Let us refer to Figure 1 in order to define these primitives formally and independently of the original predi-

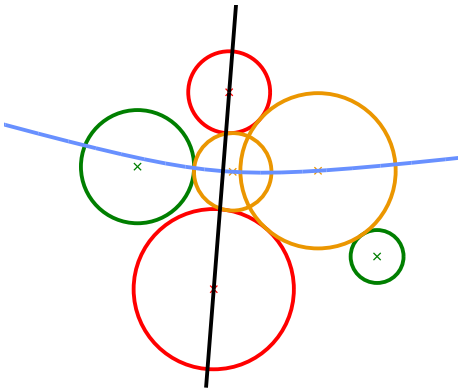

Figure 1: A typical configuration for the RadiiDifference primitive.

cate. Consider input sites B_1, B_2 (red circles, north and south in Figure 1) which define a bisector curve (blue, almost parallel to the equator). There are another two sites (green, west and south-east), each making up a triplet with B_1, B_2. Each triplet of sites corresponds to a Voronoi circle (orange, tritangent circles), whose center lies on the bisector curve. The subpredicate OrderOnBisector has to order the centers of the two Voronoi circles on the bisector. If the centers lie on opposite sides of the line joining the centers of B_1, B_2 (black line, close to vertical), the subpredicate can be decided with only Orientation tests. Otherwise, the configuration is as in Figure 1, and RadiiDifference can decide the subpredicate by comparing the radii of the two Voronoi circles.

We use an *inversion approach*, based on the following map in the complex plane: $z \mapsto 1/(z - z_0)$, where z_0 is an arbitrary fixed point. This transforms circles through z_0 to lines, while all other circles remain as such. In particular, we can choose z_0 so that a Voronoi circle is mapped to a bitangent line because, given any set of sites, one can be reduced to a point and the others will have their radii decreased, perhaps to a negative value. This transformation essentially transforms the problem of computing an additively weighted Voronoi cell to that of computing the convex hull of a set of circles. The interested reader may refer to [4] for the details of this transformation.

The RadiiDifference primitive can be reduced to deciding $\text{sign}(1/y_1 - 1/y_2)$, where y_i is a specific root of $\gamma_i y_i^2 - 2\beta_i y_i + \alpha_i = 0$, for $i = 1, 2$, and the $\alpha_i, \beta_i, \gamma_i$ are functions of the input quantities of degree 4, 5, and 6 respectively. By letting $x_i = 1/y_i$, we may concentrate on

$$f_i := \alpha_i x_i^2 - 2\beta_i x_i + \gamma_i = 0, \ i = 1, 2. \tag{2.1}$$

The problem now becomes that of ordering two specific algebraic numbers defined by the f_i (cf. [14]).

3 Algebraic preliminaries

In this section, we briefly describe the basics of multivariate resultants, Sturm theory and Descartes' rule of sign. The expressions involved are greatly simplified by the use of classical geometric invariants. For further information on resultants and invariants see [8], whereas for Sturm sequences see [24].

We start with Descartes' rule of sign. For a sequence $(\alpha_0, \ldots, \alpha_n)$ of nonzero reals, the *number of sign variations* is the number of integers $i > 0$ such that $\alpha_i \alpha_{i-1} < 0$.

Proposition 1. [8] *The number of sign variations in the sequence of the nonzero coefficients of a univariate polynomial exceeds the number of positive real roots by an even integer, possibly zero.*

Passing to resultants, consider a system of $n + 1$ polynomials in n affine variables, whose coefficients are regarded as indeterminate parameters. The *resultant* R of this system is an irreducible polynomial in the indeterminate parameters. It has integer coefficients and is well-defined up to a sign. The given system has a common root precisely when the indeterminate coefficients take values such that the resultant evaluates to zero. Different resultants exist depending on the space of the roots we wish to capture, including the *projective (or classical) resultant* and the *toric (or sparse) resultant*. They vanish respectively when there exists a common root in projective space $\mathbb{P}^n$ or in the corresponding toric variety.

Different ways of expressing R are possible, e.g., by means of a matrix determinant or a divisor of such a determinant. There are two main families of matrices, named after Sylvester (or Macaulay) on the one hand, and Bézout (or Dixon) on the other. When we are given a well-constrained system $f_1, \ldots, f_n$ in variables $x_1, \ldots, x_n$, one has to define an over-constrained system in order to apply the resultant method. One way is by adding an extra linear polynomial $f_0 := u_0 + u_1 x_1 + \cdots + u_n x_n$, as in Section 1.1. This is known as the u-resultant; its properties can be found in [8] and the references thereof.

The rest of the section discusses Sturm sequences. Given univariate polynomials $P_0, P_1 \in \mathbb{R}[x]$, their *Sturm sequence* is any (pseudo-remainder) sequence of polynomials $P_0, P_1, \ldots, P_n \in \mathbb{R}[x]$, $n \geq 1$, such that $aP_{i-1} = QP_i + bP_{i+1}$, $i = 1, \ldots, n-1$, for some $Q \in \mathbb{R}[x]$, $a, b \in \mathbb{R}$, and $ab < 0$. When a sequence is understood and $p \in \mathbb{R}$ is given, we denote by $V_P(p)$ the number

of sign variations of the sequence of values of the P_i evaluated at p.

PROPOSITION 2. *For relatively prime polynomials $A, B \in \mathbb{R}[x]$, where A is assumed square-free, consider any Sturm sequence (P_i) of $A, A'B$. Then for any $p < q$ non-roots of A, it holds that*

$$V_P(p) - V_P(q) = \sum_{A(\rho)=0,\ p<\rho<q} sign(B(\rho)).$$

The Sturm sequence here may be $(A, A'B, -A, \dots)$.

We shall apply this proposition to the polynomials defined in 2.1, namely $A(x) = f_1(x), B(x) = f_2(x)$. To simplify the computations we apply the classical geometric invariants, just as in [9], namely: $\Delta_i = \beta_i^2 - \alpha_i\gamma_i$, $i = 1, 2$, $J = \alpha_1\beta_2 - \alpha_2\beta_1$, $J' = \beta_1\gamma_2 - \beta_2\gamma_1$, $K = \alpha_1\gamma_2 + \alpha_2\gamma_1 - 2\beta_1\beta_2$. The Δ_i, K are invariant by the action of $\mathrm{SL}_2(\mathbb{C})$, and J, J' are invariant with respect to translations. $G = \alpha_1\gamma_2 - \alpha_2\gamma_1$ is not an invariant but its expression looks like one. Besides these quadratic quantities, we also use certain cubic and quartic invariants.

Supposing that f_1, f_2 have no common root and that $\alpha_i, \Delta_i \neq 0$, we have the Sturm sequence below:

$$\begin{aligned}
P_0(x) &= f_1(x)\\
P_1(x) &= f_1'(x)f_2(x)\\
P_2(x) &= -f_1(x)\\
P_3(x) &= 2\alpha_1[(2\beta_1 J - \alpha_1 G)x + (\gamma_1 J - \alpha_1 J')]\\
P_4(x) &= -\alpha_1\Delta_1(2\beta_1 J - \alpha_1 G)^2(G^2 - 4JJ')
\end{aligned}$$

If the algorithm has to consider the larger root of $f_1(x)$, it evaluates the P_i at $p = \frac{\beta_1}{\alpha_1}$ and $x \to \infty$. Then, assuming $\alpha_1 > 0$, $\mathrm{sign}(P_3(p)) = \mathrm{sign}(J)$ and computing $\mathrm{sign}(P_3(\infty))$ reduces to testing $2\beta_1 J - \alpha_1 G$. The quantity $2\beta_1 J - \alpha_1 G$ is a cubic invariant with respect to translations, whereas $R_0 := G^2 - 4JJ'$ is a quartic invariant by the action of $\mathrm{SL}_2(\mathbb{C})$. There are alternative ways to write various of the above quantities. For example, $2\beta_1 J - \alpha_1 G = -(\alpha_1 K + 2\alpha_2\Delta_1)$, whereas $G^2 - 4JJ' = K^2 - 4\Delta_1\Delta_2$. The expressions actually used in the computations are the ones that require the minimum number of operations for their computation. For example, if Δ_1 and Δ_2 are given, the expressions $\alpha_1 K + 2\alpha_2\Delta_1$ and $K^2 - 4\Delta_1\Delta_2$ are used. If this is not the case, the expressions $2\beta_1 J - \alpha_1 G$ and $G^2 - 4JJ'$ can yield a lower operation count.

4 Application of Sturm sequences

In this section, we apply Sturm theory to deciding the ordering of two specific roots, given two quadratic polynomials. We use the notation of the previous section.

case	order of roots
1	$x_2^- < x_2^+ < x_1^- < x_1^+$
2	$x_2^- < x_1^- < x_2^+ < x_1^+$
3a	$x_2^- < x_1^- < x_1^+ < x_2^+$
3b	$x_1^- < x_2^- < x_2^+ < x_1^+$
4	$x_1^- < x_2^- < x_1^+ < x_2^+$
5	$x_1^- < x_1^+ < x_2^- < x_2^+$

Table 1: The possible orderings of the roots of two quadratic polynomials and the corresponding cases.

$f_2(x_1^+)$	$f_2(x_1^-)$	$f_2'(x_1^+)$	$f_2'(x_1^-)$	J	case
$-$	$-$	any	any	any	3a
$-$	$+$	any	$-$	$+$	4
$-$	$+$	any	$+$		infeasible
$+$	$-$	$-$	any		infeasible
$+$	$-$	$+$	any	$-$	2
$+$	$+$	$-$	$-$	$+$	5
$+$	$+$	$-$	$+$		infeasible
$+$	$+$	$+$	$-$	any	3b
$+$	$+$	$+$	$+$	$-$	1

Table 2: Cases according to the first 4 signs, for $\alpha_i, \Delta_i > 0$; $\mathrm{sign}(J)$ is shown if it can be derived from the first 4 signs. The cases are as per Table 1.

In our case we always have $\Delta_i \geq 0$. Geometrically this is due to the fact that given two circles C_i and C_j no-one is contained inside the interior of the other. For a geometric interpretation of the quantities J, J', K and R_0 the interested reader may refer to [9].

By the proposition, $V_P(p) - V_P(\infty) = \mathrm{sign}(f_2(x_1^+))$ and $V_P(-\infty) - V_P(p) = \mathrm{sign}(f_2(x_1^-))$, where x_1^+ and x_1^- is, respectively, the larger and smaller root of f_1. Our algorithm uses also Sturm sequence Q of $f_1, f_1'f_2'$:

$$\begin{aligned}
Q_0(x) &= f_1(x)\\
Q_1(x) &= f_1'(x)f_2'(x)\\
Q_2(x) &= -\alpha_2[Jx + (\alpha_2\gamma_1 - \beta_1\beta_2)]\\
Q_3(x) &= 4\alpha_2\Delta_1\alpha_1^2 J^2(\alpha_1\Delta_2 + \alpha_2 K)
\end{aligned}$$

which supplies the signs of $f_2'(x_1^+), f_2'(x_1^-)$. Table 2 distinguishes the different cases based on these 4 signs. When comparing x_1^+ with x_2^+, this leads to the procedure shown in Figure 2. The shown procedure can also handle the cases where any or several of the tested quantities vanish, but this is not made explicit in Figure 2 for the sake of simplicity and readability. The evaluation tree in Figure 2 is not the only possible one. Its design, however, reflects our main aims:

1. Conclude as fast as possible (e.g., Case 5 is decided after only two comparisons, those of J and K).

This is also achieved by using certain quantities as filters for others (e.g., J' is a filter for P_4).

2. Re-use computed quantities (e.g., J can be used to compute $P_3(\infty)$ and P_4; K or J' can be used to compute P_4). This helps minimizing the expected number of operations in the tree.

3. Compute high-degree quantities as rarely as possible, i.e., the high-degree quantities should appear as low as possible in the tree (e.g., P_4, the highest-degree quantity, is a leaf of the tree).

Analogous procedures are obtained for comparing the other root pairs.

Given the tree of Figure 2, we estimate the average number of arithmetic operations for ordering x_1^+, x_2^+. Supposing that all branches are equally likely, the expected number of operations is $11\frac{1}{4}$. If each number of operations is multiplied by the maximum degree of the corresponding quantities, this count becomes $133\frac{1}{4}$. If we consider all 6 cases of Table 1 equally likely, then the expected number of arithmetic operations is $12\frac{1}{4}$. These estimates, when compared with those corresponding to the computation tree of [9], show that our Sturm-based approach is about 5% to 15% faster, on inputs that make the algorithm reach the maximum depth of the evaluation tree.

For reducing arithmetic precision, it is important to minimize the algebraic degree of the computed expressions in the input variables. By noting that $\alpha_i, \beta_i, \gamma_i$ have degree 4, 5, and 6, respectively, it is clear that the maximum degree of any computed expression is 20, determined by P_4. But if we develop the above expressions in terms of the input parameters, R_0 factorizes to two polynomials of degrees 12 and 8, respectively; this has been achieved with the Maple software package. Then, the maximum degree is 14, provided that the input is sufficiently generic for the above Sturm sequences to hold.

The procedure of Figure 2 cannot be obtained from Table 2 exactly as shown. In particular, when deciding that $K < 0$, it is not possible to exclude case 1 (case 5, respectively), provided that $J < 0$ ($J > 0$, resp.). But if we consider the information given by the resultant coefficients in conjunction with Descartes' rule, we conclude that case 1 (case 5, resp.) can be excluded; cf. Table 3. Although this shortcut does not change the evaluation procedure in this case, it illustrates how different information can be used to optimize the predicate evaluation.

4.1 Degeneracies. Due to the factorization of R_0 to two expressions of degree 12 and 8 in the input parameters respectively, the maximum algebraic degree now appears in the case of degenerate input, namely $\alpha_i = 0 \neq \alpha_j$ for $\{i, j\} = \{1, 2\}$. Geometrically this corresponds to 3 sites which have a common tangent line, or equivalently to 3 sites whose Voronoi circle has infinite radius. In this case we need quantities of degree 16 to answer our problem. It is interesting to note that these quantities do not factorize to expressions of lower degree. Recall that factorization does not commute with taking projections of polynomials modulo an ideal: here the ideal is defined by the polynomial α_i developed in terms of the input parameters. In particular, the Sturm sequence P of $f_1, f_1' f_2$, assuming $\alpha_1 = 0 < \alpha_2$, ends with $P_3(x) = 4\beta_1^2 \alpha_2 f_2(\gamma_1/(2\beta_1))$. The quantity $4\beta_1^2 f_2(\gamma_1/(2\beta_1))$ is of degree 16. Notice that the Sturm sequences cannot be obtained simply by specializing the quantities that vanish because specialization does not commute with pseudo-remaindering.

One may consider as degenerate any configuration that makes one or more of the tested quantities equal to zero. One such degenerate setting is when $\alpha_i > 0$ but $2\beta_1 J - \alpha_1 G = 0$, in which case the polynomial $P_3(x)$ is a constant. Geometrically this means that the roots of the two polynomials $f_i(x)$, $i = 1, 2$, satisfy the following relation :

$$\frac{(x_1^+ - x_2^+)(x_1^+ - x_2^-)}{(x_1^- - x_2^+)(x_1^- - x_2^-)} = -1.$$

But then, we may compute the Sturm sequence under this hypothesis and see that all tested quantities are of lower degree. In particular, P_3 becomes $2\alpha_1 \Delta_1 J'$, hence the maximum degree of any tested quantity is 11. Note that this expression for P_3 can also be obtained by

R_4	R_3	R_2	R_1	R_0	N_+	case
+	−	−	−	−	1	infeasible
+	−	−	−	+	2	infeasible
+	−	−	+	−	3	2
+	−	−	+	+	2	3
+	−	+	−	−	3	2
+	−	+	−	+	4	1
+	−	+	+	−	3	2
+	−	+	+	+	2	3
+	+	−	−	−	1	4
+	+	−	−	+	2	3
+	+	−	+	−	3	infeasible
+	+	−	+	+	2	infeasible
+	+	+	−	−	1	4
+	+	+	−	+	2	3
+	+	+	+	−	1	4
+	+	+	+	+	0	5

Table 3: Different cases according to the coefficient signs in the resultant, assuming $\alpha_i > 0$. N_+ denotes the number of positive real roots of the resultant.

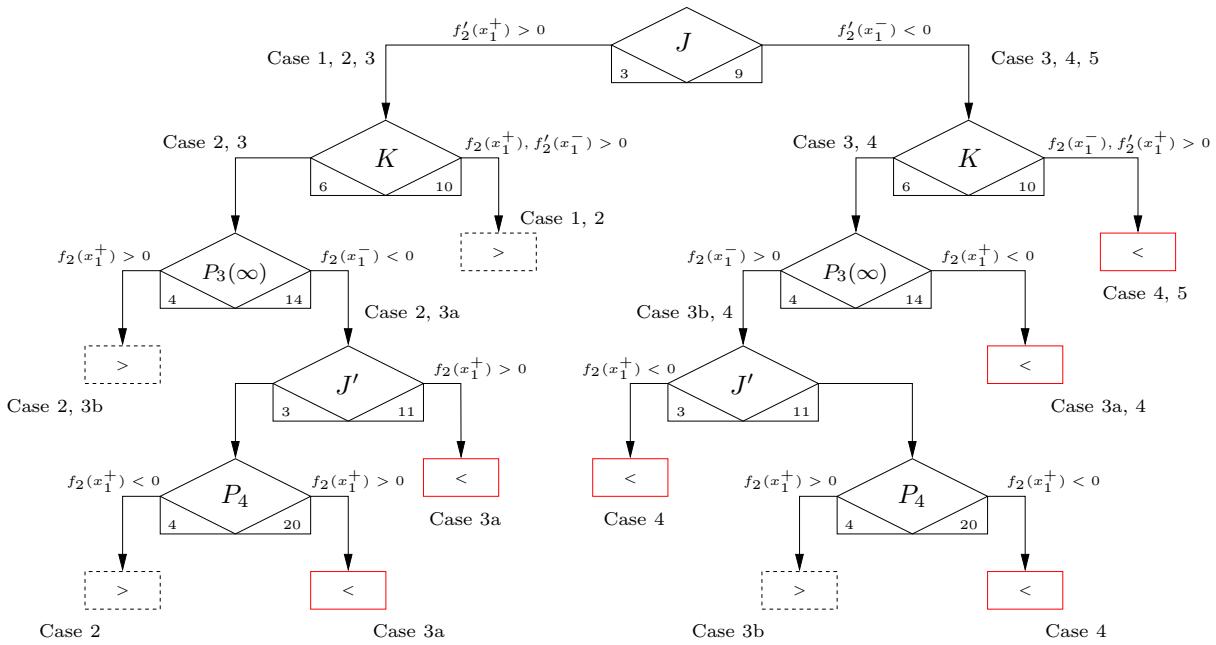

Figure 2: Evaluation procedure for deciding $x_1^+ \diamond x_2^+$, where $\diamond \in \{<, >\}$. At each tested quantity we indicate the number of operations needed in order to compute it (left) and its algebraic degree in the input (right).

simply specializing the polynomial $P_3(x)$.

We have checked that all degeneracies can be handled by the procedure of Figure 2 (and the analogous procedures for testing different pairs of roots) without modifying the shown tree substantially. More specifically, for certain nodes, the zero case can be incorporated in one of the two non-zero cases considered already. For the rest of the nodes, when the tested quantity vanishes, it is possible to conclude almost immediately and decide which case occurs. Therefore, the maximum degree of expressions tested by our algorithm becomes 16 when dealing with arbitrary inputs, including the degenerate cases. This discussion, together with that of the previous section, proves our main algorithmic result.

THEOREM 1. *It is possible to implement the algorithm of [15] for constructing the additively weighted Voronoi diagram by testing quantities of degree at most 16 in the input parameters.*

It is possible to use polynomials $f_i(y)$ for $y = 1/x$; cf. also the definition of f_i in (2.1). The maximum algebraic degree of any tested quantity in this case is 16. On the upside, we can avoid all of the above degeneracies, since $\gamma_i > 0$ by definition. This yields an alternative approach with the same maximum degree as that of Theorem 1.

5 Experimental results

We describe our code, the sample inputs, and a series of experiments that illustrate our contributions. Our code is in C++ and uses the CGAL library [7]. We have implemented caching of intermediate expressions when evaluating the predicates, which reduces significantly the number of operations. For reasons of programming simplicity, we have used the approach sketched just after Theorem 1. In our current preliminary implementation, the factorization of R_0 to polynomials of degree 12 and 8 is not undertaken. The main reason is that this requires a large number of operations; e.g., the first factor is comprised of 205 monomials in the input parameters. Thus the maximum degree of the expressions actually tested by our program is 20. We plan to conduct further experiments in order to decide whether it is worth implementing the factorization that leads to lower algebraic degree.

We present two series of experiments. The first focuses on the entire algorithm. Two data sets are considered, one random and one in almost degenerate position (Table 4). The random set consists of $N = 5 \cdot 10^5$ sites with integer coordinates uniformly distributed in the square $[-M, M] \times [-M, M]$, where $M = 10^{14}$. The weights of the sites are integers uniformly distributed in the interval $[0, R]$, where $R = 10^{11}$. About 3% of the sites are trivial. The almost degenerate data set consists

of N sites which are approximately tangent to the circle centered at the origin of radius M. The coordinates of the site centers are random integers. The radii of the sites are also random integers uniformly distributed in $[0, R]$. Less than 1% of the sites are trivial.

The algorithm used is that of the AW-Voronoi hierarchy in [15]. We use two methods for evaluating the predicates. The first assumes that the operations $\{+, -, \times, /, \sqrt{\ }\}$ are performed exactly. The second uses the procedure of Figure 2 and assumes that only the operations $\{+, -, \times\}$ are performed exactly. We refer to the two evaluation methods using the keywords *simple* and *Sturm*, respectively. We consider various number types: LEDA reals, the multiprecision floating point number `MP_Float` provided by CGAL, and the GNU multiprecision integer GMP [13]. We use the keywords `real`, `mpfloat` and `gmp` to refer to these number types, respectively. We also consider filtered versions of the above exact number types, where the filtering is dynamic and is performed via the interval arithmetic package of CGAL [22]. The built-in `double` of C++ is also used for the random data. For the almost degenerate data, `double` is insufficient, i.e., the predicates cannot be evaluated correctly.

The second series of experiments focuses on the comparison between two algebraic numbers of degree 2 (Table 5). In particular, we are given the polynomials in (2.1) and we want to compare x_1^+ and x_2^+. We use 3 different methods, namely the one that represents the roots as radicals, the method presented in [9] and our method based on the evaluation tree of Figure 2. We use the keywords *simple*, *DFMT* and *Sturm*, respectively. We consider 3 models for the bit size of the coefficients α_i, β_i and γ_i. These are $(4b, 5b, 6b)$, $(b-2, b-1, b)$ and (b, b, b), where b is a parameter. The first one corresponds to our geometric problem, where b is the bit size of the inputs of our algorithm. The second and third model correspond to a homogeneous and a generic polynomial. The number types used are the same as in the first series of experiments. The experiments using `double`s are only given for reference. We consider polynomials with randomly chosen coefficients, using two scenarios : (1) the roots of the two polynomials are entirely independent (*random* data), and (2) the larger roots of the two polynomials are equal (*degenerate* data).

All experiments were conducted on a Pentium-III architecture at 1 GHz. We used version 4.2 of LEDA and 2.4 of CGAL. The compiler used was the GNU `g++` compiler, version 2.95.3 (with options `-O2 -mcpu=pentiumpro -march=pentiumpro`). We observe that the running times of the filtered approach with random inputs are only 3 to 5 times larger than those with floating point arithmetic; this holds for all three tables. This may go against popular belief that assumes exact arithmetic to be excessively costly, when compared to numerical computation. The latter, moreover, offers no guarantee and would lead to inconsistencies when (near) degeneracies occur; in the case of AW-Voronoi diagrams, inconsistencies may appear even with certain random inputs. Our experiments thus provide another confirmation that carefully implemented exact arithmetic imposes a reasonable overhead on efficiency.

An important observation is that the filtered approach is usually at least two times faster than the non-filtered approach, and more so for almost degenerate inputs; cf. Table 4. Among the filtered methods, our Sturm-based techniques run about twice as fast as the *simple* method. It is natural that for almost degenerate inputs, the number of tests increases with filtering, because in certain cases an augmented precision is required. This increase occurs for both ORDERONBISECTOR and RADIIDIFFERENCE, and for both *simple* and *Sturm* methods, but not to the same extent. This is a manifestation of the fact that the two methods do not use the same error bounds; a larger input sample may shed light to this phenomenon.

It was expected that the number of tests should increase between random and almost degenerate inputs. It is less obvious, though, that this would not hold for the tests on P_4, the quantity examined at the maximum depth of the evaluation tree; cf. the last column of Table 4. Experimental data, not shown here, confirm that the output cases do not occur with the same frequency when the input is almost degenerate. Recall that the last 3 columns of Table 4 refer to those instances that require comparing x_1^+ and x_2^+, and different pairs of cases are distinguished by $P_3(\infty)$ and P_4. In our random data the prevailing cases are 3b ($J > 0$) and 4; hence the numerous P_4 tests. In the almost degenerate data the vast majority of cases is distributed between 2, 3b ($J < 0$) and 5. In this case $P_3(\infty)$ can decide the order of x_1^+, x_2^+ in about 99% of the cases for which K cannot yield an answer.

Another interesting corollary of our experiments is the clear improvement upon the method of [9], thanks to the procedure based on Sturm sequences, which reduces the number of operations; cf. Table 5. In addition, the inputs that do not require going to maximum depth of the evaluation tree have a lower algebraic degree, since we have replaced testing E by $P_3(\infty)$. This may have also allowed for better filtering.

Further conclusions can be drawn from Table 5 by considering the complexity as a function of bit size. We observe that, for random inputs, the cost increases in sublinear fashion, because several tests can be per-

Algorithm for Additively Weighted Voronoi Diagram using hierarchy										
	Method	Number type	Insertion time	OrderOnBisector		RadiiDifference		x_1^+-x_2^+ tree	$P_3(\infty)$	P_4
				time	# calls	time	# calls	# calls	# evals	# evals
Random	simple	`double`	60.63	1.62	50201	< 0.01	10035	n/a	n/a	n/a
		`real`	761.17	19.8		0.04				
		filter + `real`	197.68	5.08		0.04				
	Sturm	`double`	60.	1.5		0.02		8896	6198	6072
		`real`	778.59	20.32		0.12				
		`gmp`	3562.92	93.06		0.52				
		`mpfloat`	2493.55	66.54		0.97				
		filter + `real`	204.47	5.69		0.06				
		filter + `gmp`	190.15	5.35		0.08				
		filter + `mpfloat`	190.67	5.23		0.09				
Almost Degenerate	simple	`real`	3291.61	129.6	2621874	14.97	1533280	n/a	n/a	n/a
		filter + `real`	3134.17	95.72	2622036	14.32	1533426			
	Sturm	`real`	1712.26	93.64	2621874	12.31	1533280	767489	598460	825
		`gmp`	5282.56	402.89		62.04				
		`mpfloat`	3550.62	331.4		74.11				
		filter + `real`	1276.53	53.85	2622054	7.39	1533374	767523	598493	858
		filter + `gmp`	1178.41	53.1		8.18				
		filter + `mpfloat`	852.85	47.79		9.47				

Table 4: Comparison of different number types and methods for the algorithm in [15] and its predicates. Running times are in sec and refer to the total time spent in the corresponding module. "# evals" denotes number of evaluations and "n/a" refers to cases where the corresponding entry is not applicable.

formed accurately at low precision. As far as degenerate inputs are concerned, this is again true with multi-precision integer or floating-point arithmetic, which is adaptive in this sense. However, using reals implies a superlinear dependence on bit size because of lazy evaluations, which require that several evaluations be repeated as precision increases. The GMP package uses asymptotically faster algorithms than those supporting `MP_Float`. But this is not obvious from our experiments since they never treat quantities of more than about 200 bits.

6 Further work

The C++ implementation is scheduled to become part of the Cgal library. We are working on applying our approach to the 3D Voronoi problem, as well as in computing arrangements of special classes of conic arcs in the plane, such as elliptic curves.

Acknowledgments

Work partially supported by the IST Programme of the EU as a Shared-cost RTD (FET Open) Project under Contract No IST-2000-26473 (ECG - Effective Computational Geometry for Curves and Surfaces). We thank Olivier Devillers for fruitful discussions and for letting us use the implementation of the methods in [9]. We would also like to thank Monique Teillaud and Sylvain Pion for useful comments that helped us improve early versions of this paper.

References

[1] P. Angelier. *Algorithmique des Graphes de Visibilité*. PhD thesis, Université Paris VII, 2002.

[2] F. Anton, J.-D. Boissonnat, D. Mioc, and M. Yvinec. An exact predicate for the optimal construction of the additively weighted Voronoi diagram. In *Europ. Workshop Comput. Geom.*, 2002.

[3] F. Aurenhammer. Power diagrams: properties, algorithms and applications. *SIAM J. Comp.*, 16:78–96, 1987.

[4] J.-D. Boissonnat and M. I. Karavelas. On the combinatorial complexity of euclidean voronoi cells and convex hulls of d-dimensional spheres. These proceedings.

[5] J.-D. Boissonnat and F. P. Preparata. Robust plane sweep for intersecting segments. *SIAM J. Comp.*, 29(5):1401–1421, 2000.

[6] C. Burnikel. *Exact Computation of Voronoi Diagrams and Line Segment Intersections*. Ph.D thesis, Universität des Saarlandes, March 1996.

[7] *The CGAL Manual*, 2002. Release 2.4.

[8] D. Cox, J. Little, and D. O'Shea. *Using Algebraic Geometry*. Number 185 in Graduate Texts in Mathematics. Springer-Verlag, New York, 1998.

[9] O. Devillers, A. Fronville, B. Mourrain, and M. Teillaud. Algebraic methods and arithmetic filtering for exact predicates on circle arcs. *Comp. Geom: Theory & Appl., Spec. Issue*, 22:119–142, 2002.

[10] R. L. Drysdale, III and D. T. Lee. Generalized Voronoi diagrams in the plane. In *Proc. 16th Allerton Conf. Commun. Control Comput.*, pages 833–842, 1978.

Methods for number types that support $\{+, -, \times, /, \sqrt{\ }\}$											
	Degree model	b	`double`			`real`			filter + `real`		
	$(\alpha_i, \beta_i, \gamma_i)$		simple	DFMT	Sturm	simple	DFMT	Sturm	simple	DFMT	Sturm
Random	$4b$, $5b$, $6b$	4	0.37	0.28	0.24	2.42	2.82	3.7	1.23	1.06	0.86
		8	0.24	0.26	0.32	3.63	5.35	5.1	1.39	0.84	0.79
	b-2, b-1, b	25	0.28	0.23	0.24	2.41	2.95	3.73	1.28	1.1	0.81
		50	0.35	0.31	0.22	4.17	5.5	5.24	1.33	0.91	0.79
	b, b, b	25	0.35	0.27	0.25	2.51	2.96	3.66	1.39	0.83	0.79
		50	0.21	0.25	0.26	3.66	5.69	5.49	1.33	0.99	0.98
Degenerate	$4b$, $5b$, $6b$	4	0.23	0.3	0.26	114.53	122.05	111.01	117.44	125.58	113.65
		8	0.29	0.23	0.34	485.95	421.69	367.17	488.23	426.66	372.79
	b-2, b-1, b	25	0.25	0.25	0.32	127.23	128.83	128.68	130.	132.58	131.23
		50	0.44	0.23	0.42	538.78	427.07	376.2	543.46	433.61	380.87
	b, b, b	25	0.27	0.24	0.33	126.84	129.41	129.62	130.6	132.51	131.68
		50	0.38	0.23	0.27	538.71	427.24	375.95	543.66	433.86	380.48

Methods for number types that support $\{+, -, \times\}$										
	Degree model	b	`gmp`		`mpfloat`		filter + `gmp`		filter + `mpfloat`	
	$(\alpha_i, \beta_i, \gamma_i)$		DFMT	Sturm	DFMT	Sturm	DFMT	Sturm	DFMT	Sturm
Random	$4b$, $5b$, $6b$	4	25.95	23.25	11.65	9.69	0.97	0.81	0.89	0.81
		8	27.61	25.42	14.82	12.24	1.07	0.77	0.84	0.93
	b-2, b-1, b	25	25.63	23.7	12.83	9.68	0.88	0.87	0.84	0.75
		50	29.96	25.7	17.38	13.49	0.94	0.86	1.1	0.92
	b, b, b	25	26.2	23.04	12.84	10.07	1.08	0.9	0.95	0.95
		50	30.44	26.75	18.4	13.89	0.87	0.97	0.93	1.09
Degenerate	$4b$, $5b$, $6b$	4	43.16	37.68	19.42	15.78	53.33	46.26	28.53	23.58
		8	46.28	39.91	25.67	20.74	56.75	49.62	34.97	28.37
	b-2, b-1, b	25	44.01	37.9	20.54	15.59	53.8	46.88	28.7	24.37
		50	49.54	41.7	30.13	23.15	60.1	51.45	38.79	31.67
	b, b, b	25	43.82	37.8	20.53	17.03	53.24	46.71	29.83	24.57
		50	49.94	42.37	31.07	24.19	60.25	51.06	40.33	32.96

Table 5: Running times for the comparison of x_1^+ and x_2^+ considering various models and bit sizes for the coefficients. The measurements are in μsec and are the averages of 10^6 random input sequences.

[11] Z. Du, S. Pion, and C. Yap. *The CORE Library*, 1.5 edition, 2002. http://www.cs.nyu.edu/exact/core.

[12] S. Fortune. A sweepline algorithm for Voronoi diagrams. In *Proc. 2nd Annu. ACM Sympos. Comput. Geom.*, pages 313–322, 1986.

[13] T. Granlund. GMP, the GNU multiple precision arithmetic library. http://www.swox.com/gmp/.

[14] M. I. Karavelas and I. Z. Emiris. Predicates for the planar additively weighted Voronoi diagram. Technical Report ECG-TR-122201-01, INRIA, 2002. http://www.inria.fr/prisme/ECG/Results.

[15] M. I. Karavelas and M. Yvinec. Dynamic additively weighted Voronoi diagrams in 2D. In *Proc. ESA*, pages 586–598, 2002. Also INRIA Tech. Report RR-4466, INRIA Sophia-Antipolis.

[16] J. Keyser, T. Culver, D. Manocha, and S. Krishnan. MAPC: A library for efficient and exact manipulation of algebraic points and curves. In *Proc. ACM Symp. Comput. Geometry*, pages 360–369, 1999.

[17] D.-S. Kim, D. Kim, and K. Sugihara. Voronoi diagram of a circle set from Voronoi diagram of a point set: I. Topology. *CAGD*, 18:541–562, 2001.

[18] R. Klein, K. Mehlhorn, and S. Meiser. Randomized incremental construction of abstract Voronoi diagrams. *Comp. Geom: Theory & Appl.*, 3(3):157–184, 1993.

[19] G. Liotta, F. P. Preparata, and R. Tamassia. Robust proximity queries: An illustration of degree-driven algorithm design. *SIAM J. Comp.*, 28:864–889, 1999.

[20] K. Mehlhorn and S. Näher. *LEDA: A Platform for Combinatorial and Geometric Computing*. Cambridge University Press, Cambridge, UK, 2000.

[21] D. Perrucci. Personal communication, 2002.

[22] S. Pion. Interval arithmetic: An efficient implementation and an application to computational geometry. In *Workshop on Appl. of Interval Analysis to Systems and Control*, pages 99–110, 1999.

[23] M. Sharir. Intersection and closest-pair problems for a set of planar discs. *SIAM J. Comp.*, 14:448–468, 1985.

[24] C. K. Yap. *Fundamental Problems of Algorithmic Algebra*. Oxford University Press, New York, 2000.