A GENERAL APPROACH TO REMOVING DEGENERACIES *

IOANNIS Z. EMIRIS‡ AND JOHN F. CANNY‡

Abstract. We wish to increase the power of an arbitrary algorithm designed for non-degenerate input, by allowing it to execute on all inputs. We concentrate on infinitesimal symbolic perturbations that do not affect the output for inputs in general position. Otherwise, if the problem mapping is continuous, the input and output space topology are at least as coarse as the real euclidean one and the output space is connected, then our perturbations make the algorithm produce an output arbitrarily close or identical to the correct one. For a special class of algorithms, which includes several important algorithms in computational geometry, we describe a deterministic method that requires no symbolic computation. Ignoring polylogarithmic factors, this method increases only the worst-case bit complexity by a multiplicative factor which is linear in the dimension of the geometric space. For general algorithms, a randomized scheme with arbitrarily high probability of success is proposed; the bit complexity is then bounded by a small-degree polynomial in the original worst-case complexity. In addition to being simpler than previous ones, these are the first efficient perturbation methods.

Key words. Input degeneracy, ill-conditioned problems, symbolic perturbation, infinitesimals, randomization, determinants, roots of polynomials, algorithmic complexity

AMS subject classifications. 68Q10, 68Q20, 68Q25, 68U05

1. Introduction.

Quite often algorithms are designed under the assumption of input non-degeneracy. Although they can have many specific forms, most degeneracies in geometric or algebraic algorithms reduce to a division by zero, or to a sign determination for a value which is zero. In this article we describe efficient methods for systematically removing such degeneracies using symbolic infinitesimal perturbations. Our methods apply to every algorithm that can be implemented on a real RAM.

This work is influenced by the treatment of the problem in [12] and, in a more general context, [21]. The main contribution of this article is to introduce the first general and efficient perturbations from the viewpoint of worst-case complexity. Previous methods incurred an extra computational cost that was exponential in some parameter of the input size.

The principal domains of applicability are geometric and algebraic algorithms over an infinite ordered field. Take, for instance, a Convex Hull algorithm in arbitrary dimension over the reals. It is typically described under the hypothesis of general position which excludes several possible instances, such as more than k points lying on the same $(k-1)$-dimensional hyperplane, in m-dimensional euclidean space, $m \geq k$. For an algebraic algorithm, consider Gaussian Elimination without pivoting that works under the hypothesis that the pivot never vanishes. Our perturbation scheme accepts a program written under this hypothesis and outputs a slightly longer program that works for all inputs.

The perturbations introduced change the original input instance into a non-degenerate one which is arbitrarily close, in the usual euclidean metric, to the original input. For algorithms that branch only on the sign of determinants, which includes several important geometric algorithms, we propose a deterministic method. It increases the worst-case algebraic complexity of the algorithm by a multiplicative factor

* Supported by a David and Lucile Packard Foundation Fellowship and by NSF Presidential Young Investigator Grant IRI-8958577.

‡ Computer Science Division, University of California at Berkeley, Berkeley, CA 94720.

of $O(\log d)$ and its worst-case bit complexity by a factor of $O(d^{1+\alpha})$, where d is the dimension of the geometric space of the input objects and α is an arbitrarily small positive constant accounting for the polylogarithmic factor. In addition to its efficiency, this scheme is easy to implement, which makes it attractive for practical use [14]. The perturbation, although defined in terms of a symbolic infinitesimal variable, does not require any symbolic computation.

For general algorithms, we propose a randomized scheme that incurs a factor of $O(D^{1+\alpha})$ on the algebraic complexity, where D is the highest total degree in the input variables of any polynomial in the program. Under the bit model, the worst-case running time of the new program is asymptotically bounded by $\phi^{3+\alpha}$, where ϕ is the original bit complexity of the original one; in both cases α denotes an arbitrarily small positive constant.

All claims about the efficiency of our approach are based on worst-case complexities. Yet there exist other measures, such as output size, under which the perturbations cause a much more significant increase in running time. For a degenerate input the output may be of constant size while, under the perturbation, the given algorithm cannot do significantly better than what its worst-case performance predicts.

The next section defines the computational model, formalizes the use of infinitesimals as well as the notion of degeneracy, and specifies the problem at hand. Section 3 is a comparative study of previous work on handling degeneracies. Sections 4 and 5 describe the perturbations for algorithms that branch on determinants and on arbitrary rational functions respectively; each section includes an application. We conclude with a summary and a discussion of directions for further work.

2. Preliminaries.

2.1. Model of computation. Our results hold for any infinite ordered field, yet we present them in terms of the reals $\mathbb{R}$. We choose the real-arithmetic Random Access Machine (RAM) as our model; it is described in [18] and is a more powerful version of the simple RAM defined in [1].

An *input* of size N consists of a finite real vector $\mathbf{x} = (x_1, \ldots, x_N)$ and a particular input instance is $\mathbf{a} = (a_1, \ldots, a_N) \in \mathbb{R}^N$. The real RAM can perform real arithmetic exactly with respect to the four basic operations in $\{+, -, *, /\}$ and can branch on the sign of a rational function in the input variables, evaluated at the particular input instance. The machine can also write to and read from a memory that can store an arbitrary number of exact real values. The chosen model abstracts certain issues that may arise in practice, such as real number representation and exactness of arithmetic operations.

The set of instructions that implements a given algorithm on the real RAM forms a *program*; no program can alter itself. The subset of instructions executed on some input instance, forms an *execution path*. For a specific program on the real RAM and a given input, the *output* is unique and is expressed by a finite real vector, possibly describing other structures such as graphs.

In Section 5 we shall require an extension of the model, namely that there exists an explicit finite integer bound D on the total degree in $\mathbf{x}$ of any polynomial computed in the course of the program. The concept of this bound appears in the Machine of [2] and in the algebraic decision tree of [18].

Under the *algebraic model*, the worst-case complexity of an algorithm equals the maximum number of arithmetic operations, branching and memory access instructions

executed on any input. More realistically, we may wish to consider the effect of the operands' bit size on the speed of arithmetic operations. Under the *bit model* there is a cost function on each instruction of the program and the worst-case complexity of an algorithm equals the maximum sum of the costs of all instructions on the execution path corresponding to any input. In both models, the time to access the memory is assumed constant and therefore does not affect the total asymptotic running time. Branches also take constant time, provided that each number carries an extra bit indicating whether it equals zero or not. Without this extra bit the branching cost would be linear in the size of the operand but this would not affect the results in this article.

The cost of arithmetic operations depends on the particular operation executed as well as the bit size of the operands. For integers of size b, addition and subtraction have cost $O(b)$, while the cost of multiplication and division, due to an algorithm by Schönhage and Strassen is $O(b \log b \log\log b)$, [1]. For rationals, the Greatest Common Divisor (GCD) is factored out at every arithmetic operation, and finding it takes $O(b \log^2 b \log\log b)$ time, [1]. Let $M(b) = O(b \log^2 b \log\log b)$ bound the bit complexity of any operation on two rational numbers, each represented by a pair of $O(b)$-bit integers. We define the bit size of a rational number to be the maximum bit size of the numerator and denominator.

Given an input instance, our perturbations define a new instance in terms of a symbolic variable that is never evaluated, which implies that instead of real numbers the program may have to manipulate polynomials in this variable. Formally, this can be thought of as producing a new program with the same control flow in which every arithmetic and branching instruction is substituted by a black box that implements the appropriate operation on univariate polynomials. Of course, the algebraic as well as the bit cost associated with each instruction changes.

2.2. Infinitesimals. Our approach in removing degeneracies is to add to the input values arbitrarily small quantities. To this effect we make use of infinitesimals. The process of extending the field of reals by an infinitesimal is a classical technique, formalized in [3], and used by the second author in [4].

DEFINITION 2.1. We call ϵ *infinitesimal* with respect to $\mathbb{R}$ if the extension $\mathbb{R}(\epsilon)$ is ordered so that ϵ is positive but smaller than any positive element of $\mathbb{R}$. Clearly, the sign of any polynomial in ϵ is the sign of the non-zero term of lowest degree.

Alternatively, it is enough for ϵ to belong to the reals and take a sufficiently small positive value so that it avoids the roots of a finite set of polynomials; we shall see that this is the set of all polynomials appearing at a branch in the real RAM program. The smallest positive root in any of these zero sets is larger than some positive real ϵ_0, hence it suffices that $\epsilon = \epsilon_0$. This idea may be seen as a special case of the "Transfer Principle" [20].

An immediate consequence is that symbolic perturbations of the input by ϵ-polynomials are equivalent to defining a new real instance by setting ϵ equal to ϵ_0. Then the execution path on perturbed input is that of some real input which implies that the algorithm halts on perturbed input provided that it does for all real inputs.

2.3. Degeneracy. Before formalizing the notion of degeneracy, we examine it with respect to some concrete problems. For the Matrix Inversion problem an intrinsically degenerate input is a singular matrix, for which the output is undefined. An input degeneracy may depend not only on the particular problem but also on the

algorithm. An algorithm-induced degeneracy for the Gaussian Elimination algorithm without pivoting arises at a matrix with a singular principal minor.

Yap in [21] uses the Convex Hull problem in the plane to distinguish between intrinsic and algorithm-induced degeneracies. Assume that in the output space topology polytopes of distinct combinatorial structure lie in disjoint components. Then three collinear points constitute an intrinsic degeneracy because the mapping of point sets to convex hulls is not continuous. On the other hand, two covertical points have nothing special with respect to the mapping of point sets to their convex hull. They may, however, constitute a degeneracy with respect to a particular algorithm that solves the problem by using a vertical sweep-line or relies on some vertical partitioning of the plane.

We formalize the discussion by considering both input and output spaces as real topological subspaces of finite dimension.

DEFINITION 2.2. A *problem mapping* associates with almost every input instance exactly one (exact) solution.

DEFINITION 2.3. The input instances on which the problem mapping is not defined or not continuous form the set of *intrinsic* degeneracies for this problem.

An algorithm and, equivalently, the respective real RAM program that compute a problem mapping typically impose certain restrictions on the input instances. Hence the need to consider the mapping defined by a specific algorithm.

DEFINITION 2.4. An *algorithm mapping* is defined by a particular real RAM program and is a restriction of the problem mapping to exclude at least all intrinsic degeneracies.

In what follows no distinction is made between an algorithm and the real RAM program that implements it.

DEFINITION 2.5. The *input space* of an algorithm mapping is a real space of finite dimension. In the context of the computational model, defined in Section 2.1, the dimension equals N. The *output space* of an algorithm mapping is a topological space, equal to the union of the disjoint finite-dimensional real topological spaces associated with the distinct execution paths of the real RAM program. These output subspaces will be called *leaf subspaces*.

This terminology reflects the fact that branching causes the program to have a tree structure. The problem and algorithm output spaces can be either connected or disconnected. Usually, a disconnected output space can be made connected by identifying points in different leaf subspaces. Then the overall space inherits the topology of the leaf subspaces with no open sets intersecting two leaf subspaces.

This is possible in the example of the Convex Hull problem mapping, where polytopes which are identical as point sets but lie in different subspaces can be identified, thus producing a connected output space. Yet, the problem mapping remains discontinuous. There exist other topologies that will make this space connected and the problem mapping continuous. One example is the metric topology where the distance of two polytopes is measured by the volume of their symmetric difference.

DEFINITION 2.6. An input instance is *degenerate* with respect to some algorithm if, during its execution, it causes some branch rational function f, whose numerator and denominator polynomials are not identically zero, either to be undefined or to evaluate to zero while the algorithm produces no solution for the case $f = 0$. Equivalently, the input instance is in *general position* or *generic* if there is no such test function f.

Clearly, the domain of an algorithm mapping is precisely the set of all generic inputs, which excludes *all* degeneracies, i.e. both intrinsic and induced ones. An important class of degenerate inputs are those that lead a program to division by zero. This case is included in the previous definition by requiring that the program is robust enough to have a zero test on the denominator before each division.

DEFINITION 2.7. The set of *induced degeneracies* includes exactly those degenerate inputs that are not intrinsic degeneracies.

The input space can be partitioned into equivalent classes, where each instance produces the same sign sequence on the branch rational functions reached during execution. The classes that do not make any branch function f, as specified in Definition 2.6, vanish or be undefined, partition the domain of the algorithm mapping into cells of input instances that produce an output instance in the same leaf subspace. The union of inputs that cause some branch polynomial to vanish contains the degenerate subset and has positive codimension since it is the finite union of polynomial zero sets. Hence the degenerate subset has positive codimension which agrees with the informal view of degeneracies as special cases or events of zero probability.

2.4. Problem definition. Given is an algorithm that solves a problem under the hypothesis of non-degeneracy. Our aim is, given an arbitrary input instance $\mathbf{a} = (a_1, \ldots, a_N)$, to define in a systematic way some other instance so that the same algorithm can always produce a meaningful output. The new instance, denoted $\mathbf{a}(\epsilon) = (a_1(\epsilon), \ldots, a_N(\epsilon))$, will be defined by adding to each a_i a polynomial in some symbolic positive infinitesimal ϵ. The discussion in Sections 2.1 and 2.2 implies that $\mathbf{a}(\epsilon)$ can be given as input to the same real RAM program.

To provide some intuition on the desired effects of perturbations we return first to the Matrix Inversion problem. Given a perturbed singular matrix, (i) the algorithm should return its inverse and (ii) the perturbed input should be arbitrarily close to the original one under the standard euclidean metric. Condition (i) will follow from the correctness of the algorithm on generic inputs once we establish that perturbed matrices are nonsingular. Restricted to nonsingular matrices, the problem mapping is continuous. On a perturbed nonsingular matrix (ii) implies that the output is arbitrarily close to the exact solution; to recover the latter we can simply set the infinitesimal to zero in the perturbed output.

For the Convex Hull problem, there is always a solution, its combinatorial nature however may prohibit the problem mapping from being continuous. Take, for instance, the volume of the symmetric difference between the actual output and the exact convex hull as the distance between these two polytopes. This volume tends to zero with ϵ since the induced metric topology is at least as coarse as the euclidean one. Under this metric the output space is connected and the approximate solution is arbitrarily close to the exact one.

We now specify the desired properties of a perturbation in terms of the outputs obtained under different circumstances. Limits are understood with respect to the topology of the input and output spaces.

DEFINITION 2.8. Given an input instance $\mathbf{a}$, a *strongly valid perturbation* defines a new instance $\mathbf{a}(\epsilon)$ which lies in general position, tends to $\mathbf{a}$ as ϵ approaches zero and satisfies the following conditions:

- If $\mathbf{a}$ is in general position, the algorithm produces the same output whether it runs on $\mathbf{a}$ or it runs on $\mathbf{a}(\epsilon)$ and at the end ϵ is set to 0.

- If $\mathbf{a}$ is an induced degeneracy, the output space is connected with topology not finer than the euclidean one and the problem mapping is continuous, then the algorithm on $\mathbf{a}(\epsilon)$ returns an output that either produces the exact solution by setting $\epsilon = 0$ or tends to the exact solution in the limit as $\epsilon \to 0$.
- If $\mathbf{a}$ is degenerate and some hypothesis of the previous case fails, then the algorithm produces a correct solution for $\mathbf{a}(\epsilon)$.

A more practical definition describes requirements for a weaker perturbation in terms of the input space.

DEFINITION 2.9. Given an input instance $\mathbf{a}$, a *valid perturbation* defines a non-degenerate instance $\mathbf{a}(\epsilon)$ which tends to $\mathbf{a}$ as ϵ approaches zero, such that when $\mathbf{a}$ is in general position all branches take the same direction on $\mathbf{a}(\epsilon)$ as on $\mathbf{a}$.

PROPOSITION 2.10. *Suppose that the input space and leaf subspace topologies are at least as coarse as the real euclidean topology. Then any valid perturbation is strongly valid.*

Proof. It suffices to show that the three conditions of Definition 2.8 are satisfied. For generic inputs all branches take the same direction on $\mathbf{a}$ and $\mathbf{a}(\epsilon)$, hence the two outputs lie in the same leaf subspace. No ϵ-term can be the most significant in any polynomial in the output, because this is the output on a generic instance. Therefore all such terms can be ignored by setting $\epsilon = 0$, thus obtaining the exact solution.

In the second case, since we deal with an induced degeneracy, an exact solution exists. Whenever the problem mapping is continuous the algorithm mapping is also continuous in its own domain. Since the input space topology is sufficiently coarse, the fact that $\mathbf{a}(\epsilon)$ tends to $\mathbf{a}$ and the continuity property imply that the output produced on perturbed input tends to the exact solution as $\epsilon \to 0$. In the more favorable cases, the exact output is recovered by setting $\epsilon = 0$.

The correctness of the algorithm on generic inputs and the hypothesis that $\mathbf{a}(\epsilon)$ is generic imply that the last condition is satisfied. □

In what follows we focus to problems that satisfy the hypothesis of the previous proposition, hence reducing the validity requirements to those of Definition 2.9. Furthermore, we consider perturbations of the following form:

$$a_i(\epsilon) = a_i + \epsilon\, c_i,$$

for $c_i \in \mathbb{Z}$ independent of a_i and ϵ. The post-processing necessary to recover the exact answer is usually a very case-specific process. We discuss the case of convex hulls at the end of Section 4 and also refer the reader to [21], [12] and [8].

3. Other Work. The most naive approach is to handle each special case separately, which is tedious for implementors and unattractive for theoreticians. Random perturbations are frequently alluded to and one such scheme is studied in this article. Their main feature is that they trade randomness for efficiency.

Symmetry breaking rules in Linear Programming are the earliest systematic approaches to the problem. Dantzig presents such a method in [6] which relies on an infinitesimal ϵ. Consider a Linear Program reduced to finding non-negative values for the $m+n$ variables x_j, such that the sum of all slack variables $\sum_{j>n}^{m+n} x_j$ is minimized. The perturbation consists in adding a power of ϵ to every non-negative constant b_i, where $1 \le i \le m$:

$$\sum_{j=1}^{n} a_{i,j} x_j + x_{n+i} = b_i + \epsilon^i,$$

This forces the perturbed constants to be strictly positive and eliminates the degenerate case of having $b_i = 0$, for some i in $\{1, \ldots, m\}$.

Edelsbrunner and Mücke systematize in [12] a scheme called Simulation of Simplicity (SoS for short), already presented in [9], [11], [13] and [10]. It applies to algorithms that accept n input objects, each specified by d parameters, and whose tests are determinants in the nd parameters, just as our deterministic perturbation (1) of the next section. SoS perturbs every input parameter $p_{i,j}$ into

$$p_{i,j}(\epsilon) = p_{i,j} + \epsilon^{2^{i\delta - j}},$$

where $\delta > d$ and ϵ is a symbolic infinitesimal; this is a valid scheme under our definition. The sign of the perturbed determinant is the sign of the smallest-degree term in its ϵ-expression and can be calculated numerically.

Finding the sign of the perturbed determinant is, on the average, pretty fast. In the worst case however, the determinant computation takes $\Omega(2^d)$ steps, since it may have to check that many minors of the perturbed matrix. This bound is obtained by calculating the number of distinct vectors $(v_1, \ldots, v_{d-2})$, where d denotes the order of the original matrix. Every v_i is a positive integer less than or equal to d and for every $i < j$, $v_i \leq v_j$. In [12] every such vector is associated with a distinct minor that may have to be evaluated. This analysis pertains to Λ matrices, to be defined in the next section. Matrices of the second kind, the Δ matrices, require more steps in the worst case, for the same order. In short, SoS incurs a worst-case exponential overhead in d.

Yap in [21] provides a more general framework, which includes SoS as a special case, where branching occurs at arbitrary rational functions. His technique is consistent relative to infinitesimal perturbations [22] and valid; here we examine it as applied to polynomial tests. Let $PP = PP(x_1, \ldots, x_n)$ denote the set of all power products of the form

$$w \;=\; \prod_{i=1}^{n} x_i^{e_i}, \quad e_i \geq 0$$

in the n input variables. A total ordering $\leq_A$ on PP is *admissible* if, for all $w, w', w'' \in PP$,

$$1 \leq_A w \qquad \text{and} \qquad w \leq_A w' \Rightarrow ww'' \leq_A w'w''.$$

Let $w_1, w_2, \ldots$ be the ordered list of power products larger than 1, i.e. those with at least one positive exponent. Then, each polynomial $f(\mathbf{x})$ is associated with the infinite list

$$S(f) \;=\; (f, f_{w_1}, f_{w_2}, \ldots)$$

where f_{w_k} is the partial derivative of f with respect to w_k; for example, $f_{x_2^2 x_3} = \partial^3 f / (\partial^2 x_2 \, \partial x_3)$. The sign of a non-zero polynomial f is the sign of the first polynomial in $S(f)$ whose value at the actual input is not zero, which can always be found after examining a finite number of terms.

Yap focuses on sparse n-variate polynomials, with m denoting the maximum degree of any variable. In the case that all variables are of degree m, a polynomial f has $(m+1)^n - 1 \geq m^n$ non-trivial derivatives. On the average, only a few partial

derivatives will have to be evaluated, but at worst, all of them have to be computed and the complexity is $\Omega(m^n)$.

Dobrindt, Mehlhorn and Yvinec [8] studied the problem of intersecting an arbitrary polytope with a convex one in three dimensions, proposed an efficient perturbation and discussed post-processing in this context. The interesting feature of their technique is that it controls the direction of perturbation. In particular, since the facet structure is given, the polytope vertices are forced to be perturbed outward.

In a slightly different vein, Canny used a *structural* perturbation in [4] to ensure that the input semi-algebraic sets are in general position. One immediate application is to motion-planning algorithms, where these sets describe obstacles or prohibited space. The perturbation preserves emptiness and number of connected components of the original sets by using sequences or towers of infinitesimals.

Perturbation methods have been applied in other cases to eliminate degeneracies with respect to particular problems, as in [17] for instance.

Lastly, Emiris and Canny in [14] extend the applicability of the deterministic perturbation introduced in this article to another two geometric branching tests, most importantly to the InSphere test. They also propose a new variant of the scheme that eliminates the polynomial factor in the asymptotic bit complexity overhead with respect to the two tests examined here.

4. Branching on determinants. We first restrict attention to algorithms whose branching depends exclusively on the sign of determinants in the input variables, which is the case with several geometric algorithms. We concentrate on two specific types of determinants that cover important algorithms, such as those computing Convex Hulls and Hyperplane Arrangements. Our approach can be applied to other types of determinants too, as demonstrated in [14].

Assume that the input parameters represent n input objects $p_1, p_2, \ldots p_n$, each specified by d parameters. Without loss of generality, each $p_i = (p_{i,1}, p_{i,2}, \ldots p_{i,d})$, is a point in $\mathbb{R}^d$. We are interested in the case that the dimension d is arbitrary, whence the total input size is nd. It is also reasonable to assume that a constant fraction of the points are distinct, thus establishing a lower bound on the parameters' bit size. In practice this condition can be guaranteed by using an initial check to eliminate duplicate points; in most settings the complexity of this phase is dominated by that of the main algorithm.

We perturb deterministically every parameter $p_{i,j}$ to obtain $p_{i,j}(\epsilon)$, where ϵ is an infinitesimal symbolic variable.

$$p_{i,j}(\epsilon) = p_{i,j} + \epsilon\,(i^j). \tag{1}$$

Although ϵ is never assigned a real value, we shall show that no symbolic computation is necessary.

First consider matrix Λ_{d+1} whose rows correspond to points $p_{i_1}, p_{i_2}, \ldots, p_{i_{d+1}}$:

$$\Lambda_{d+1} = \begin{bmatrix} 1 & p_{i_1,1} & p_{i_1,2} & \cdots & p_{i_1,d} \\ 1 & p_{i_2,1} & p_{i_2,2} & \cdots & p_{i_2,d} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & p_{i_{d+1},1} & p_{i_{d+1},2} & \cdots & p_{i_{d+1},d} \end{bmatrix}.$$

Testing the sign of this determinant comes up in various contexts. We call it the Sidedness test because, given a query point $p_{i_{d+1}}$ and the hyperplane spanned by the

other d points, the sign of the determinant indicates on which side of the hyperplane the query point lies. The determinant vanishes if and only if the query point lies on the hyperplane; the positive and negative side of the hyperplane are determined by the order of the d points defining it. This test is sometimes called the Orientation test, since it may be regarded as deciding the relative orientation of the $d+1$ points in the sense of [12]. In fact, the column of ones should be rightmost, but it is a constant-time operation to obtain the orientation of the points from the sign of $det\Lambda_{d+1}$.

Let us refer to the new matrix that contains the corresponding perturbed parameters as the perturbed matrix, denoted by $\Lambda_{d+1}(\epsilon)$. The modified program that runs on perturbed input will be computing the sign of its determinant, which is given by the following expression.

$$det\Lambda_{d+1}(\epsilon) \;=\; det\Lambda_{d+1} + \;(\epsilon^k \; terms, 1 \leq k \leq d-1) + \;\epsilon^d \begin{vmatrix} 1 & i_1 & i_1^2 & \dots & i_1^d \\ 1 & i_2 & i_2^2 & \dots & i_2^d \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & i_{d+1} & i_{d+1}^2 & \dots & i_{d+1}^d \end{vmatrix},$$

where the last term is the determinant of V_{d+1}, a $(d+1)\times(d+1)$ Vandermonde matrix, with

$$detV_{d+1} = \prod_{\substack{k,l \,\in\, \{1,\dots,d\} \\ k \,>\, l}} (i_k - i_l).$$

The second matrix of interest has rows representing input points p_{i_1}, p_{i_2}, ... p_{i_d}:

$$\Delta_d = \begin{bmatrix} p_{i_1,1} & p_{i_1,2} & \dots & p_{i_1,d} \\ p_{i_2,1} & p_{i_2,2} & \dots & p_{i_2,d} \\ \vdots & \vdots & & \vdots \\ p_{i_d,1} & p_{i_d,2} & \dots & p_{i_d,d} \end{bmatrix}.$$

This test decides the orientation of points expressed in their homogeneous coordinates. In a dual setting, such as in [11], the input objects are hyperplanes in $(d-1)$-dimensional space and the test indicates on which side of the first hyperplane lies the intersection of the other $d-1$ hyperplanes. Then, the determinant vanishes if and only if the d hyperplanes have a non-empty intersection; for this reason, this is called the Transversality test.

The corresponding matrix $\Delta_d(\epsilon)$ of the perturbed parameters has determinant

$$det\Delta_d(\epsilon) \;=\; det\Delta_d \;+\; (\epsilon^k \; terms, 1 \leq k \leq d-1) \;+\; \epsilon^d \begin{vmatrix} i_1 & i_1^2 & \dots & i_1^d \\ i_2 & i_2^2 & \dots & i_2^d \\ \vdots & \vdots & & \vdots \\ i_d & i_d^2 & \dots & i_d^d \end{vmatrix},$$

where the coefficient of ϵ^d is the determinant of another Vandermonde matrix U_d and can be expressed as follows:

$$detU_d \;=\; \prod_{k=1}^{d} i_k \;\; detV_d \;=\; \prod_{k=1}^{d} i_k \quad \prod_{\substack{k,\,l\,=\,1 \\ k \,>\, l}}^{d} (i_k - i_l).$$

LEMMA 4.1. *Given a real RAM program, there exists a positive real constant ϵ_0 such that, for every positive real $\epsilon < \epsilon_0$, every $\Lambda_{d+1}(\epsilon)$ and $\Delta_d(\epsilon)$ matrix occurring at a branching node of the program is nonsingular and its determinant has constant sign.*

Proof. The perturbed determinants are univariate polynomials over the reals. Any polynomial over an ordered field that is not identically zero has an algebraic set of roots of positive codimension thus, for univariate polynomials, this set is the union of a finite number of points.

Neither $det\Lambda_{d+1}(\epsilon)$ nor $det\Delta_d(\epsilon)$ are identically zero because the highest-order term never vanishes, since all indices are distinct and positive. Hence, there exists a finite number of roots for each symbolic determinant in ϵ. Letting ϵ_0 be the minimum positive such root over all test determinants in the program proves the lemma. □

THEOREM 4.2. *Perturbation (1) is valid with respect to algorithms that branch on determinants of $\Lambda_{\delta+1}$ and Δ_δ, for $\delta \leq d$, where d is the dimension of the geometric space in which the input points lie.*

Proof. The perturbed instance is clearly arbitrarily close to the original one since ϵ tends to zero. This instance is also in general position because both kinds of perturbed determinants have a well-defined sign that is never zero for sufficiently small ϵ, from the previous lemma. Finally, since the sign of perturbed determinants is the sign of the lowest order non-vanishing term, when the original determinant is non-zero, it dominates the ϵ-polynomial. Then all branches take the same direction, for a given instance, before and after the perturbation. □

We now address the question of computing the sign of the perturbed determinant. One obvious way is to evaluate the terms in the determinant's ϵ-expansion in increasing order of the exponent of ϵ. The process stops at the first non-vanishing term and reports its sign. This is essentially the approach adopted by SoS and Yap's technique. Our perturbation scheme lends itself to a more efficient trick which reduces the determinant calculation to a characteristic polynomial computation, which also avoids the requirement for symbolic manipulation.

$$det\Lambda_{d+1}(\epsilon) = \frac{1}{\epsilon}\begin{vmatrix} \epsilon & p_{i_1,1}(\epsilon) & \dots & p_{i_1,d}(\epsilon) \\ \vdots & \vdots & & \vdots \\ \epsilon & p_{i_{d+1},1}(\epsilon) & \dots & p_{i_{d+1},d}(\epsilon) \end{vmatrix} =$$

$$= \frac{1}{\epsilon} det\left(\begin{bmatrix} 0 & p_{i_1,1} & \dots & p_{i_1,d} \\ \vdots & \vdots & & \vdots \\ 0 & p_{i_{d+1},1} & \dots & p_{i_{d+1},d} \end{bmatrix} + \epsilon\, V_{d+1}\right) = \frac{1}{\epsilon} det\,(L + \epsilon\, V_{d+1}).$$

Having implicitly defined L, denoting by I_k the $k \times k$ unit matrix and relying on the fact that every Vandermonde matrix is invertible, we have

$$det\Lambda_{d+1}(\epsilon) = \frac{1}{\epsilon} det(-V_{d+1})\, det(-(V_{d+1})^{-1} L - \epsilon\, I_{d+1}) =$$

$$= \frac{1}{\epsilon}(-1)^{d+1}\, detV_{d+1}\, det(M - \epsilon\, I_{d+1}). \tag{2}$$

Similarly,

$$det\Delta_d(\epsilon) = det(\Delta_d + \epsilon U_d) = det(-U_d)\, det(-(U_d)^{-1}\Delta_d - \epsilon I_d) =$$

$$= (-1)^d \prod_{k=1}^{d} i_k\, detV_d\, det(N - \epsilon\, I_d).$$

Matrices M and N are defined implicitly. Notice that we have reduced the computation of a symbolic determinant to calculating the characteristic polynomial in ϵ of M or N respectively.

We now prove the efficiency of this approach. Let $MM(k)$ denote the number of multiplications and divisions needed to multiply two $k \times k$ matrices, which is currently $O(k^{2.376})$ [5].

LEMMA 4.3. *Computing the sign of perturbed determinants* $\det\Lambda_{d+1}(\epsilon)$ *and* $\det\Delta_d(\epsilon)$ *can be done in* $\mathcal{O}(MM(d)\log d)$ *arithmetic steps.*

Proof. The determinant and the inverse of a $d \times d$ Vandermonde matrix takes at most $O(d^2)$ arithmetic steps [23], while computing M or N as a matrix product takes $O(MM(d))$ operations. Computing $det(M - \epsilon\, I_{d+1})$ or $det(N - \epsilon\, I_d)$ is a characteristic polynomial computation for which there exists an algorithm by Keller-Gehrig [16] requiring $O(MM(d)\log d)$ operations. This algorithm is purely numeric as it transforms matrix M or N respectively to a new matrix that contains the coefficients of the characteristic polynomial in the last column. □

A brief discussion of modular arithmetic is in order here because, besides being the most commonly used method to carry out exact arithmetic on computers, it is also the most economical for computing the perturbed determinants. Let k denote the total number of finite fields required for a particular computation, which is proportional to the bit size of the quantity that is to be eventually computed. Suppose that each finite field $\mathbb{Z}_q$ is defined by a constant-size prime integer q which can be obtained in constant time from an existing and sufficiently long list of primes. Following the exposition in [1], the *first stage* consists of mapping each matrix element into its k residues, the *second stage* performs the particular computation in k different finite fields and the *third stage* applies the Chinese Remainder Theorem to find the answer from its k residues. The first and third stages have both bit complexity $O(M(k)\log k)$ while that of the second stage depends on the computation performed. The modular method is applicable to rational inputs with the same asymptotic complexity [7].

Let s be the maximum bit size of any input parameter with $s = \Omega(\log n)$, since we have assumed that a constant fraction of input points are distinct.

THEOREM 4.4. *Consider algorithms that branch on the determinants of* $\Lambda_{\delta+1}$ *and* Δ_δ, *for* $\delta \leq d$, *where* d *is the dimension of the geometric space of the input points. Perturbation (1) increases the asymptotic running-time complexity of the algorithm under the algebraic model by* $\mathrm{O}(\log d)$. *Under the bit model, the worst-case complexity is increased by a factor of* $\mathrm{O}(d^{1+\alpha})$, *where* α *is an arbitrarily small positive constant that accounts for the polylogarithmic factors.*

Proof. The previous lemma proves the claim on the algebraic complexity since the original complexity of computing a $d \times d$ determinant is $\Theta(MM(d))$ [15].

In what follows we concentrate without loss of generality to the Sidedness test. In the original setting, the worst-case bit size of the determinant is $\Theta(ds)$ and using modular arithmetic requires $k = \Theta(ds)$ distinct finite fields. The first stage maps d^2 quantities to their respective residues, the second stage computes the determinant modulo some prime q, while the last stage's complexity is dominated. Hence the overall worst-case complexity is

$$\Theta(d^2(ds)^{1+\alpha} + dsMM(d)),$$

where α is an arbitrarily small positive constant.

For the perturbed determinant we must compute the coefficients of the characteristic polynomial of M. Observe from (2) that this is a scalar multiple of the ϵ-polynomial $\det \Lambda_{d+1}(\epsilon)$, therefore the latter's coefficient sizes provide upper bounds on the sizes of the characteristic polynomial coefficients. Now, each entry of $\Lambda_{d+1}(\epsilon)$ is a sum of an original point coordinate and a perturbation quantity, hence its bit size is the maximum of s and $d \log n$. All coefficients of $\det \Lambda_{d+1}(\epsilon)$ are sums of determinants of order at most d that have entries of bit size $s + d \log n$. Thus the coefficients have size $O(ds + d^2 \log n)$, which also provides an asymptotic upper bound on the number of finite fields. Hence the new bit complexity is

$$O(d^2(ds + d^2 \log n)^{1+\alpha} + (ds + d^2 \log n) MM(d) \log d),$$

where α is another arbitrarily small positive constant. To complete the proof we apply the asymptotic lower bound on s. □

The section concludes with an application to the Beneath-Beyond Convex Hull algorithm for general dimension, presented in [10]. The algorithm is incremental and relies on the hypothesis of general position with respect to the two tests used for branching: The first simply sorts the points along some coordinate assuming that no two points have the same coordinate. The second is essentially the Sidedness test, called once the convex hull of a subset of the points is constructed: Given a $(d-1)$-dimensional facet and a query point, decide whether the point lies on the same side of the facet as the hull or not. Under perturbation (1) this test can be implemented by at most two Sidedness tests.

The perturbation is transparent with respect to the rest of the algorithm. The two branching tests can be thought of as subroutines that are given subsets of input points and return a non-zero sign, in order to avoid degeneracies. The polytope constructed is simplicial, which means that all faces are simplices. Restricting attention to the facets, we note that their number is not minimum, because some may be created in order to subdivide a non-simplex facet into simplices, while others may include input points in the interior of some facet which have been perturbed into polytope vertices.

Our approach is most favorable in applications where such redundancy is immaterial, for instance in computing the volume of the convex hull. In this case, under the symmetric volume metric, the problem mapping is continuous and the exact answer is readily obtained by setting $\epsilon = 0$ in the expression of each partial volume that makes up the overall polytope. These partial volumes are $\Lambda_{d+1}(\epsilon)$ determinants computed in the course of the algorithm, so there is no extra cost for calculating the exact volume.

The artificial facets may have to be eliminated for other applications through a post-processing phase. This involves checking every facet against every one of its d adjacent facets by computing a Λ_{d+1} determinant. If it vanishes, then certain $(d-2)$-dimensional faces must be eliminated and, eventually, certain input points may have to be removed from the vertex set of the convex hull. The algebraic complexity of this phase is asymptotically equal to the product of $dMM(d)$ and the number of facets in the approximate output, hence its bit complexity is dominated by that of the algorithm.

5. Branching on Arbitrary Rational Functions. For the general case where the branching tests are arbitrary rational functions, we propose a randomized perturbation which is easy to implement and applies to algebraic problems such as Matrix

Inversion and Linear Programming as well as geometric algorithms whose branching functions are not covered by those examined above.

Let f be an arbitrary rational function whose sign determines the direction taken at some branch and express it as p/q, where p, q are polynomials in the input variables, each of total degree bounded by D; recall that D is the maximum total degree in the input variables of any polynomial in the real RAM program. Suppose that the input variable vector $\mathbf{x}$ belongs to $\mathbb{R}^n$ and let $\mathbf{a} = (a_1, \ldots, a_n)$ be a particular input instance, hence the input size is n.

For a given input, define the perturbed instance $\mathbf{a}(\epsilon) = (a_1(\epsilon), \ldots, a_n(\epsilon))$ as follows:

$$a_i(\epsilon) = a_i + \epsilon\, r_i \tag{3}$$

where ϵ is an infinitesimal symbolic variable and r_i is a random integer.

Each r_i is chosen uniformly over a range that depends on the desired probability that none of the branching polynomials vanishes. This probability of success can be fixed to be arbitrarily high. It is parameterized by a real constant $c \geq 1$; all claims in this section hold with probability at least $1 - 1/c$.

The total number of polynomials appearing at the numerator or denominator of a branch expression is at most $2 \cdot 3^T$, where T is the maximum number of branches on any execution path. Schwartz's lemma [19] requires that the range of the random values contains at least as many values as the product of c and the total degree of the polynomial whose roots we wish to avoid. Here, this polynomial is the product of all branch polynomials, hence its degree is bounded by $2 \cdot 3^T D$. Therefore, the bit size of the perturbation quantities is

$$\lceil \lg c + \lg D + (\lg 3)\, T + 1 \rceil,$$

where lg denotes the logarithm of base 2.

It is feasible that for some set of random variables, $\mathbf{a}(\epsilon)$ will still cause some branching polynomial to vanish. In this case the perturbation has failed, so the algorithm is restarted and new random variables are picked, independently and uniformly distributed over the same range. It is not clear that any deterministic scheme could avoid the zeros of all polynomials without taking time at least exponential in the number of variables. Intuitively, our method is faster because it randomly selects one n-dimensional perturbation vector instead of trying out all possible ones.

LEMMA 5.1. *Let the entries of* $\mathbf{r} = (r_1, \ldots, r_n)$ *be independently and uniformly chosen integers of* $\lceil \lg c + \lg D + (\lg 3)\, T + 1 \rceil$ *bits each, for any* $c \geq 1$. *Then, there exists with probability at least* $1 - 1/c$, *a positive real constant* ϵ_0 *such that, for every positive real* $\epsilon < \epsilon_0$, *every branching rational function* $f(\mathbf{a} + \epsilon\mathbf{r})$ *is defined, non-zero and of constant sign.*

Proof. Let $g(\mathbf{a}+\epsilon\mathbf{r})$ be any polynomial appearing at the numerator or denominator of some branch expression and let $G(\mathbf{a}+\epsilon\mathbf{r})$ be the product of all distinct polynomials g. By hypothesis, none of these polynomials is identically zero, therefore G also is not identically zero. For a moment, fix $\epsilon = 1$ and consider $G(\mathbf{a} + 1\mathbf{r})$ as a polynomial in $\mathbf{r}$, whose degree in $\mathbf{x}$ and $\mathbf{r}$ is the same. Since D bounds the total degree of any polynomial g, the total degree of G is at most $2 \cdot 3^T D$. Now we apply a lemma proven in [19]. The probability that $\mathbf{r}$, chosen uniformly at random with the given size, is a

root of $G(\mathbf{a}+1\mathbf{r})$ is at most $1/c$. All claims that follow concern the particular $\mathbf{r}$ and hold with probability at least $1-1/c$.

First observe that none of the polynomials $g(\mathbf{a}+1\mathbf{r})$ vanishes at $\mathbf{r}$, hence every $g(\mathbf{a}+\epsilon\mathbf{r})$ may be regarded as a polynomial in ϵ that is not identically zero. Consequently, its zero set is of positive codimension and, more specifically, a finite point set. Consider the minimum positive root for every g and let ϵ_0 be the minimum over all polynomials g. □

THEOREM 5.2. *Perturbation (3) is valid with arbitrarily high probability, with respect to any algorithm that branches on rational functions in the input variables.*

Proof. The perturbed instance is arbitrarily close to the original one as ϵ tends to zero. Branches decide on the sign of a perturbed rational expression, which is the sign of the lowest-order term in the ϵ-polynomial that does not vanish. By the previous lemma all polynomials have a constant non-zero sign for sufficiently small ϵ, hence $\mathbf{a}(\epsilon)$ is in general position. For non-degenerate inputs all query polynomials have a non-vanishing real part, i.e. a term independent of ϵ which dominates the sign. □

What is the tradeoff in efficiency? The algebraic complexity of the algorithm is increased by the time required to manipulate the ϵ-polynomials symbolically which depends on D, since the degree of every polynomial in ϵ is the same as its total degree in $\mathbf{x}$. The bit complexity is also affected by D but not by c which is fixed.

LEMMA 5.3. *Under perturbation (3) the algebraic time complexity of the branching instructions and the arithmetic operations is* $\mathrm{O}(D)$ *and* $\mathrm{O}(D\log^2 D)$ *respectively.*

Proof. Each operation in $\{+,-,\times,/\}$ involves multiplication of ϵ-polynomials and a Greatest Common Divisor computation to reduce to lowest terms so that the degree bound D is observed. The multiplication takes time $O(D\log D)$ and the GCD $O(D\log^2 D)$, [1]. Branching instructions must find the lowest non-vanishing term in the corresponding ϵ-polynomial, which takes $O(D)$ time. □

Degree D cannot be bounded in general by a polynomial in the algebraic complexity, which implies that the perturbation may be prohibitively expensive under the algebraic model. Exponentiating a rational number, for instance, takes roughly a logarithmic number of steps in the exponent, while on perturbed input the worst-case algebraic complexity is at least linear in it, which means the complexity overhead is exponential in the original complexity. However, we obtain better bounds by considering bit complexities.

THEOREM 5.4. *Under the algebraic model, the running-time increases due to perturbation (3) by a multiplicative factor of* $\mathrm{O}(D^{1+\alpha})$*, where D is the maximum total degree in the input variables of any polynomial in the real RAM program and α is an arbitrarily small positive constant accounting for the polylogarithmic factor. Under the bit model, the overhead for the worst-case complexity is* $\mathrm{O}(\phi^{2+\alpha}(n,s))$*, where $\phi(n,s)$ asymptotically bounds the original worst-case bit complexity of the algorithm, s is the maximum bit size of the input quantities and α is an arbitrarily small positive constant.*

Proof. By the previous lemma for every instruction the overhead is $O(D\log^2 D)$. This establishes the algebraic complexity overhead.

By the definition of T there exists an execution path with bit complexity $\Omega(T)$, hence $\phi(n,s)=\Omega(T)$. By the definition of D, there exists a path where a polynomial of degree D in the input variables is computed. Since the only legal arithmetic operations lie in $\{+,-,*,/\}$, there must exist an earlier operation on this path computing a polynomial of degree at least $D/2$, hence computing values of bit size $sD/2$. The operation that uses these values as operands has bit cost $\Omega(sD/2)=\Omega(D)$. Hence

$\phi(n,s) = \Omega(D)$.

After the perturbation, the same program operates on perturbed quantities; their starting bit size is multiplied by $O(\log D + T)$, assuming c is constant. Moreover, the algebraic complexity has overhead $O(D \log^2 D)$. Hence, the worst-case bit complexity overhead is

$$O((\log D + T) D \log^2 D). \tag{4}$$

Recalling the two lower bounds on $\phi(n,s)$, we have $\phi(n,s)^{2+\alpha} = \Omega(TD^{1+\alpha})$, which bounds the bit complexity overhead (4), for some appropriate $\alpha > 0$. □

COROLLARY 5.5. *Perturbation (3) does not affect the worst-case bit complexity class of the algorithm. In particular, if the original complexity lies in P or EXPTIME, then the complexity on perturbed input also lies in P or EXPTIME respectively.*

Proof. Immediate from the previous theorem. □

Taking up the running example of Gaussian Elimination for the Matrix Inversion problem, we observe that no checks for zero denominators have to be carried out on perturbed input. Perturbation thus eliminates the need for interchanging rows. Computation is symbolic, with GCD operations at every step in order to cancel common terms and thus prohibit the degree in ϵ of the symbolic polynomials from growing exponentially in the number of examined rows. For nonsingular matrices, the result is obtained by setting ϵ to zero at the end. For singular instances, this causes some denominator to vanish, so we take the limit as ϵ goes to zero. For singular matrices the result is some real matrix that approximates, in a sense, the inverse.

6. Conclusion. We studied algorithms modeled as programs on real Random Access Machines with inputs from an infinite ordered field and described perturbations on the input, such that an algorithm designed under the assumption of non-degeneracy can be applied to all inputs. Our perturbations satisfy the validity condition set out in Section 2 which guarantees the relevance of the output with respect to the initial problem.

We defined a deterministic method for algorithms with determinant tests and a randomized one for arbitrary test functions. The first applies to algorithms from computational geometry whose branching tests can be expressed as a determinant of a Λ or Δ matrix. Ignoring polylogarithmic factors in the geometric dimension, the deterministic scheme does not affect the algebraic complexity but incurs an overhead to the worst case bit complexity that is linear in dimension. The second perturbation, applicable to most geometric and algebraic algorithms, incurs a worst-case overhead under the bit model that is bounded by a small-degree polynomial in the original complexity. Both methods are characterized by their conceptual simplicity and are significantly faster than previous ones.

Examining branching tests that come up in other geometric algorithms and trying to improve on efficiency are natural extensions to this work, partly fulfilled in [14]. It is also interesting to attempt extending the notion of degeneracy over finite fields, where the lack of order makes our definition of degeneracy invalid. Another direction of generalization is to observe that each leaf subspace is associated with a semi-algebraic set defined by the branch polynomials on the respective execution paths. We may wish to perturb these sets into general position.

Acknowledgment. We wish to thank K. Mehlhorn for several useful comments.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft and J. D. Ullman, *The Design and Analysis of Computer Algorithms,* Addison-Wesley, Reading, MA, 1974.

[2] L. Blum, M. Shub and S. Smale, *On a theory of computation and complexity over the real numbers:* NP*-completeness, recursive functions and universal machines,* Bull. Amer. Math. Soc., 21 (1989), pp. 1–46.

[3] J. Bochnak, M. Coste and M. F. Roy, *Géométrie Algébrique Réelle,* Ergebnisse der Mathematik 3, No. 12, Springer-Verlag, Berlin, 1987.

[4] J. F. Canny, *Computing roadmaps of semi-algebraic sets,* Proc. 9th Symp. on Applied Algebra, Algebraic Algorithms and Error-Corr. Codes, Lecture Notes in Computer Science, No. 539, Springer-Verlag, Berlin, 1991, pp. 94–107.

[5] D. Coppersmith and S. Winograd, *Matrix multiplication via arithmetic progressions,* J. Symb. Comput., 9 (1990), pp. 251–280.

[6] G. B. Dantzig, *Linear Programming and Extensions,* Princeton University Press, Princeton, 1963.

[7] J. H. Davenport, Y. Siret and E. Tournier, *Computer Algebra*, Academic Press, London, 1988.

[8] K. Dobrindt, K. Mehlhorn and M. Yvinec, *A complete framework for the intersection of a general polyhedron with a convex one,* Proc. 3rd Workshop Algorithms Data Struct., Lecture Notes in Computer Science, No. 709, Springer-Verlag, Berlin, 1993, pp. 314–324.

[9] H. Edelsbrunner, *Edge-skeletons in arrangements with applications,* Algorithmica, 1 (1986), pp. 93–109.

[10] ———, *Algorithms in Combinatorial Geometry,* Springer-Verlag, Heidelberg, 1987.

[11] H. Edelsbrunner and L. J. Guibas, *Topologically sweeping an arrangement,* Proc. 18th ACM Symp. on Theory of Computing, 1986, pp. 389–403.

[12] H. Edelsbrunner and E. P. Mücke, *Simulation of simplicity: A technique to cope with degenerate cases in geometric algorithms,* ACM Trans. Graphics, 9 (1990), pp. 67–104.

[13] H. Edelsbrunner and R. Waupotitsch, *Computing a ham-sandwich cut in two dimensions,* J. Symb. Comput., 2 (1986), pp. 171–178.

[14] I. Emiris and J. Canny, *An efficient approach to removing geometric degeneracies,* Proc. 8th ACM Symp. on Computational Geometry, 1992, pp. 74–82.

[15] J. von zur Gathen, *Algebraic complexity theory,* Annual Review of Computer Science, No. 3, J. Traub ed., Annual Reviews, Palo Alto, 1988, pp. 317–347.

[16] W. Keller-Gehrig, *Fast algorithms for the characteristic polynomial,* Theor. Comp. Sci., 36 (1985), pp. 309–317.

[17] C. Monma, M. Paterson, S. Suri and F. Yao, *Computing euclidean maximum spanning trees,* Proc. 4th ACM Symp. on Computational Geometry, 1988, pp. 241–251.

[18] F. P. Preparata and M. I. Shamos, *Computational Geometry,* Springer-Verlag, New York, 1985.

[19] J. T. Schwartz, *Fast probabilistic algorithms for verification of polynomial identities,* J. ACM, 27 (1980), pp. 701–717.

[20] A. Tarski, *A Decision Method for Elementary Algebra and Geometry,* University of California Press, Berkeley, 1948.

[21] C.-K. Yap, *Symbolic treatment of geometric degeneracies,* J. Symb. Comput., 10 (1990), pp. 349–370.

[22] ———, *A geometric consistency theorem for a symbolic perturbation scheme,* J. Comp. Sys. Sci., 40 (1990), pp. 2–18.

[23] R. Zippel, *Interpolating polynomials from their values,* J. Symb. Comput., 9 (1990), pp. 375–403.