A General Approach to Removing Degeneracies *

Ioannis Emiris John Canny

Computer Science Division
University of California, Berkeley
Berkeley CA 94720

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 direct perturbations that do not affect the output for inputs in general position. Otherwise, given the continuity of the problem mapping, our perturbations cause the algorithm to produce an output arbitrarily close, if not coincident, to the correct one. For a special class of algorithms that includes most geometric ones, we describe a deterministic method that affects only the bit complexity, by incurring an extra factor, polynomial in the input size. For general algorithms, a randomized scheme is applied with arbitrarily high probability of success. Then, the bit complexity is bounded by a small-degree polynomial in the original complexity. In addition to being simpler than previous ones, these are the first efficient perturbation methods.

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 paper we describe efficient methods for systematically avoiding such degeneracies using symbolic perturbations. Our methods apply to algorithms that can be represented as *algebraic branching programs*, which includes most calculations performed with real number inputs.

Symbolic perturbations have been studied in [10] and in a more general context in [18]. The main contribution of this paper is to introduce the first general and efficient direct perturbations. Previous methods incurred an extra computational cost that was exponential in the input size.

The principal domains of applicability are to geometric algorithms and algebraic algorithms over the real numbers. Take, for instance, a Convex Hull algorithm in arbitrary dimension. It is typically described under the hypothesis of general position which excludes several possibilities, such as more than k points lying on the same $(k-1)$-dimensional hyperplane, in k-dimensional euclidean space. A more detailed study of geometric algorithms appears in [12].

For an algebraic problem, consider Gaussian Elimination, which is most simply implemented under the hypothesis that the pivot never vanishes. Our perturbation scheme accepts an algebraic branching program written under these hypotheses, and outputs a slightly longer program that works for all inputs.

Direct perturbations change the original input instance into a non-degenerate one, which is arbitrarily close (in the usual euclidean metric) to the original input. We give a deterministic method for algorithms that branch only on the sign of determinants, which includes many or most geometric algorithms. This does not affect the algebraic complexity of the algorithm but increases its bit complexity by a factor of $\mathcal{O}(d^{2+\alpha})$, where d is the dimension of the geometric space of the input objects and α is an arbitrarily small positive constant. In several cases this factor is constant.

For general algorithms, we propose a randomized scheme. This incurs a factor of $\mathcal{O}(d^{1+\alpha})$ on the algebraic complexity, where d is the highest degree in the input variables of any polynomial in the program. Under the bit model, the running time of the new program is asymptotically smaller than $\phi^{3+\alpha}$, where ϕ is the running time of the original one; α is again an arbitrarily small positive constant.

The next section provides all definitions. Section 3 is a comparative study of previous work on handling degeneracies. Sections 4.1 and 4.2 describe the

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

CH3062-7/91/0000/0405$01.00 © 1991 IEEE

direct perturbations for algorithms with determinant and rational tests respectively. We conclude with a discussion of future directions in our work.

2 Preliminaries

2.1 Computation model

We are interested in algorithms that perform arithmetic operations and can branch. The *input* is a finite set of numeric values from some ordered infinite field. We focus on the field of reals, although our results apply to any ordered infinite field. Let $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbf{R}^n$ be the input variables and $\mathbf{a} = (a_1, \ldots, a_n)$ an actual input instance.

Based on the real RAM model of [15] and influenced by [1] and [14], we define an *algebraic branching program* to be a (finite) directed graph G together with a function Φ. The edges represent control flow. The program can write to and read from a memory bank of real numbers $y_1, y_2, \ldots, y_m$, where m is bounded by a polynomial in n. Every program may produce different outputs; each *output* is a set of finite objects, uniquely defined by a subset of the input and memory bank variables.

The vertices of G have zero, one or three outgoing edges. There exists exactly one vertex of outdegree one (*input vertex*). It has no incoming edge and is the vertex where execution begins. Function Φ assigns

- to every vertex v of outdegree one (*operation vertex*) an instruction of the form

  $$f_v = f_{v_1} \circ f_{v_2} \text{ or } f_v = c \circ f_{v_1},$$

  where, for $i \in \{1, 2\}$, f_{v_i} either belongs to $\{x_1, \ldots, x_n\}$ or is some y_j whose value has been computed at some vertex u that belongs to every path leading to v; $\circ \in \{+, -, \times, /\}$ and c is a real constant; the computed value of f_v is stored at some y_k;

- to every vertex v of outdegree three (*branching vertex*) a test of the form

  $$f_w : 0$$

  where f_w belongs to $\{x_1, \ldots, x_n\}$ or is some y_j with value computed at some vertex w which lies on every path from the input vertex to v; branching depends on whether f_w is positive, zero or negative;

- to every vertex of outdegree zero (*output vertex*) an output.

Given input $\mathbf{a} \in \mathbf{R}^n$, the program will traverse a *computation* path $P(\mathbf{a})$ in G from the source to some output vertex. Branching depends on the sign of the rational expression in the input variables at the specific branching vertex. The program is essentially a flowchart; the data-flow graph is embedded and has structure that depends on the results of the branching.

We shall be interested in two ways of assigning a cost to a specific computation. The first supposes that all operations have unit cost, therefore the length of $P(\mathbf{a})$ expresses the cost of the algebraic computation.

More realistically, we may wish to consider the effect of the operands' bit size on the speed of arithmetic operations. We shall say that under the *bit model* of the algebraic branching program, there is a cost function on the vertices of the program. Branching, input and output vertices have unit cost. The cost of operation vertices depends on the particular operation executed as well as the bit size of the operands. If this size is $\mathcal{O}(b)$, then addition and subtraction have cost $\mathcal{O}(b)$, while the cost of multiplication and division is denoted by $M(b)$. There exists an algorithm by Schönhage and Strassen which obtains $M(b) = \mathcal{O}(b \log b \log\log b)$ [13]. The total cost of a computation equals the sum of the costs over all vertices on $P(\mathbf{a})$. Under both viewpoints, the time to access the memory is assumed constant and therefore does not affect the total running time asymptotically.

Our perturbations will be defined in terms of a symbolic variable ϵ, which implies that computation on the perturbed input is symbolic. This formally corresponds to modifying the given program into a new one represented by the same graph, but where f_v and y_j are polynomials in ϵ, for each vertex v and every j in $[1, \ldots, m]$. Then the cost of operation and branching vertices is higher.

An equivalent view is to construct a new program by expanding every operation vertex into a set of numeric operations that compute the polynomial. Similarly, every branching vertex corresponds to a sequence of branching vertices, each testing one term of the ϵ-polynomial. The new program has the same type of vertices as the original one but is longer. Its complexity depends on the maximum path length from the source to some output vertex.

An additional provision is that of exact arithmetic. We regard our perturbation scheme as built on top of an algorithm which works correctly for non-degenerate input and thus can handle round-off errors; see [18] for a relevant discussion.

2.2 Degeneracy

The notion of degeneracy is hard to formalize because it may depend not only on the particular problem but also on the algorithm chosen to attack it. For example, an *intrinsic* or *problem-dependent* de-

generacy is a singular matrix, in the context of the Matrix Inversion problem. It constitutes an input for which there is no output. An *algorithm-dependent* degeneracy for the Gaussian Elimination algorithm that solves the Inversion problem for non-singular matrices is a matrix with a singular principal minor.

Yap in [18] uses the Convex Hull problem to distinguish between, what he calls, inherent and algorithm induced degeneracies. An example of the former kind, in the planar case, is for 3 points to be collinear. On the other hand, 2 covertical points have nothing special with respect to the problem mapping but may constitute a degeneracy for a particular algorithm that uses, say, a vertical sweep-line or relies on some vertical partitioning.

The occurrence of these degenerate configurations, as well as many others, can be formulated as polynomials evaluating to zero. This paper focuses on algorithm-dependent degeneracies, which subsume intrinsic ones, and are characterized as follows.

Definition 2.1 *An input instance is* degenerate *with respect to some algorithm if it causes the rational expression f at some branching vertex of the algorithm's algebraic branching program to evaluate to zero. Equivalently,* $\mathbf{a} \in \mathbf{R}^n$ *is in* general position *if there is no test rational expression f such that $f \not\equiv 0$ and $f(\mathbf{a}) = 0$.*

Let us adopt a more abstract standpoint and consider the space of all input instances. It can be embedded in $\mathbf{R}^n$, equipped with the standard euclidean metric. Every test expression may be regarded as a mapping of input space to $\{+, 0, -\}$. The preimage of $\{0\}$ under a particular expression is a set in $\mathbf{R}^n$ with codimension at least 1. The union of all these preimages partitions the input space into cells that lead to the same output.

Definition 2.2 *A* problem mapping *associates with almost every input instance a unique output, which is called an (exact)* solution.

The solution may be thought of as the output produced by a solver of infinite computational power. Intrinsic degeneracies are exactly the points at which this mapping fails to exist or is not continuous. Each belongs to the preimage of $\{0\}$ under the mapping of some test expression.

An algorithm mapping is a restriction of the problem mapping to instances that are not intrinsic degeneracies. Algorithm-dependent degeneracies may also be viewed as the union of the preimages of $\{0\}$ under the mappings of all test expressions.

2.3 Problem definition

Given is an algorithm that solves a problem under the assumption of non-degeneracy. Our aim is to directly perturb an arbitrary input instance $\mathbf{a}$ into some other instance $\mathbf{a}(\epsilon)$ in a systematic way so that the same algorithm can always produce a meaningful output. The perturbation is controlled by the infinitesimal variable ϵ. A *valid direct perturbation* must define $\mathbf{a}(\epsilon)$ in general position and satisfy the following condition:

- For non-degenerate inputs, the algorithm produces the same output whether it runs on $\mathbf{a}$ or $\mathbf{a}(\epsilon)$.
- For degenerate inputs, if the problem mapping is continuous, then the algorithm on $\mathbf{a}(\epsilon)$ returns either the exact solution or an output arbitrarily close to it.
- Otherwise, the algorithm produces a correct solution for $\mathbf{a}(\epsilon)$ and the latter is arbitrarily close to $\mathbf{a}$.

In the first case, it suffices to prove that every test expression, regarded as an ϵ-polynomial, has a non-vanishing constant term. In other words, that the paths followed in the program for $\mathbf{a}$ and $\mathbf{a}(\epsilon)$ are identical. Then the final output that is obtained by ignoring all ϵ-terms is the same as the one produced on $\mathbf{a}$.

In the second case, note that the algorithm mapping is continuous in its own domain. Refer to the cells defined by the preimages of $\{0\}$, each corresponding to a unique output. Observe that the output space is a topological subspace of a real space, defined by the algorithm mapping. Then, the continuity property together with the proximity of the inputs imply the proximity of the respective outputs. Hence, to demonstrate the validity of the perturbation, it suffices to show the proximity of $\mathbf{a}$ and $\mathbf{a}(\epsilon)$ in the topology in which the problem mapping is continuous. Valid perturbations are most powerful in the special case in which it makes sense to let the infinitesimal approach zero; this directly produces the exact solution.

In summary, all we need to prove for the validity of our direct perturbations is the following condition: that each defines a non-degenerate instance $\mathbf{a}(\epsilon)$ arbitrarily close to $\mathbf{a}$, such that when the latter is in general position, $P(\mathbf{a})$ and $P(\mathbf{a}(\epsilon))$ are identical.

To illustrate, consider the Matrix Inversion problem, for which there is no solution for an unperturbed singular matrix. In this case we only require that the algorithm returns a solution to the perturbed instance which has to be arbitrarily close to the original one, under the standard euclidean metric.

Restricted to non-singular matrices, the problem mapping is continuous. Then a valid perturbation produces an output arbitrarily close to the correct one. In fact, after obtaining the output on the perturbed input, we can let the infinitesimal variable go to zero and arrive at the exact solution.

For the Convex Hull problem, there is always a solution. Its combinatorial nature prohibits the output space topology from being continuous. Point sets that lie arbitrarily close in input space may yield convex hulls with different number of vertices, which thus belong to disconnected regions of output space. Nonetheless, there exist measures of success in output space, under which the approximate solution is arbitrarily close to the exact one. Take for instance the volume of the symmetric difference between the actual output and the exact convex hull. This volume tends to zero with the infinitesimal variable.

Specific applications may require that the facets of the hull include exactly the points that define them, which implies that extra (degenerate) points should be removed. Then a post-processing phase is necessary in order to produce the solution. Edelsbrunner in [9] discusses the issue of post-processing.

2.4 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 [2], and used by the second author in [3] and [5].

Definition 2.3 *We call* ϵ infinitesimal *with respect to* $\mathbf{R}$ *if the extension* $\mathbf{R}(\epsilon)$ *is ordered so that* ϵ *is positive but smaller than any positive element of* $\mathbf{R}$.

We write $0 < \epsilon \ll 1$ to indicate that every polynomial in ϵ has value smaller than 1. The sign of a polynomial in ϵ equals the sign of the non-zero term of lowest degree in ϵ. Towers of field extensions by an infinitesimal are also possible. When extending $\mathbf{R}(\delta)$ recursively by ϵ we write $0 < \epsilon \ll \delta \ll 1$ to indicate that no polynomial in ϵ can exceed δ.

Alternatively, ϵ can belong to the reals and assume an arbitrarily small positive value. Under the algebraic branching program model it is immaterial whether we view ϵ as an infinitesimal or an arbitrarily small real. To see this, consider a special case of the "Transfer Principle" [17]. There is a finite number of polynomials encountered at branching vertices on $P(\mathbf{a})$, so there is a minimum positive real value among all of their roots. As long as ϵ is smaller than that minimum, it may take any positive value and none of these polynomials will change sign. Hence, the algorithm will follow the same path and, in general, it will behave in the same way as if ϵ was an infinitesimal extension. In this paper, we make use of both standpoints and regard ϵ sometimes as an extension to the reals and sometimes as an arbitrarily small real number.

The general strategy is to transform the real input values into ϵ-polynomials. But can we still run the same algorithm on symbolic input? Do we know that it shall halt? The answer is affirmative. Due to the above argument, the path traced on the perturbed input will be the same as if we had substituted ϵ by a sufficiently small positive real value. Thus, there exists an actual real input for which the algorithm behaves the same as for the perturbed input.

3 Other approaches

Symmetry breaking rules in linear programming, exemplified by [6] and [7], are the earliest systematic approaches to our problem. Edelsbrunner and Mücke generalize in [10] a scheme called Simulation of Simplicity (SoS for short), presented in [8], [11] and [9]. SoS applies to algorithms whose tests are determinants, just as our deterministic direct perturbation. It introduces the following perturbation of input parameter $p_{i,j}$:

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

where $\delta \geq d$ and d is the dimension of the geometric space in which the input objects lie. The sign of the perturbed determinant is the sign of the smallest-degree term in its ϵ-expression.

In the worst case, this computation takes $\Omega(2^d)$ steps since it may have to check that many determinants of submatrices 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 [10] every such vector is associated with a distinct submatrix whose determinant may have to be evaluated.

Yap in [18] deals with the general setting where the test polynomials are arbitrary and shows, in [19], that his scheme is consistent relative to infinitesimal perturbations. First it imposes a total ordering on all power products

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

where $\mathbf{a} = (a_1, \ldots, a_n)$ is the input. 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 $p(\mathbf{a})$ is associated with the infinite list

$$S(p) = (p, p_{w_1}, p_{w_2}, \ldots)$$

where p_{w_k} is the partial derivative of p with respect to w_k. The sign of a non-zero polynomial p is taken to be the sign of the first polynomial in $S(p)$ whose value is

not zero, which can always be found after examining a finite number of terms.

The time complexity to evaluate an n-variate s-term polynomial p is proportional to ns. Yap focuses on sparse polynomials for which $s \leq d$, where d denotes the maximum degree of any variable in p. In the case that all variables are of degree d, p has at least d^n non-trivial derivatives. Their subsequent evaluation requires $\Omega(ns')$ multiplications for each derivative of s' terms. At worst, all partial derivatives have to be computed and d is exponential in n. Then the complexity is $\Omega(d^n)$, which is exponential in the input size.

In short, both methods incur at least an exponential worst-case complexity overhead under the algebraic branching program model. Our methods guarantee a polynomial overhead in addition to being significantly simpler.

4 Direct perturbations

This section contains the main contributions of the paper. It describes valid perturbations to cope with degeneracies. First we examine algorithms whose branching decisions depend on the sign of determinants and offer a deterministic method. Allowing arbitrary test polynomials leads us to a randomized method. In both cases we perturb directly the input objects by an amount controlled by an infinitesimal variable.

4.1 Determinant tests

We first restrict attention to algorithms whose test expressions are determinants of two specific types, to be specified below. The importance of the case stems from the fact that it covers several computational geometry algorithms, including those computing convex hulls and hyperplane arrangements; [12] expands on this topic.

Under the algebraic program model, degeneracy occurs exactly when a test determinant vanishes. Let the n distinct input objects be $p_1, p_2, \ldots p_n$. Every p_i is defined by d real parameters $p_{i,1}, p_{i,2}, \ldots p_{i,d}$, for an arbitrary integer d. We can think of the input as n points in $\mathbf{R}^d$. We perturb every parameter $p_{i,j}$ to obtain $p_{i,j}(\epsilon)$ where ϵ is a symbolic variable.

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

All computations on the perturbed input are carried out symbolically.

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

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

The perturbed matrix $\Lambda_{d+1}(\epsilon)$ contains the corresponding perturbed parameters. Its determinant is given by

$$\begin{aligned} det\Lambda_{d+1}(\epsilon) &= det\Lambda_{d+1} \\ &+ (\epsilon^k \ terms, 1 \leq k \leq d-1) \\ &+ \epsilon^d det V_{d+1}, \end{aligned}$$

where V_{d+1} is a $(d+1) \times (d+1)$ Vandermonde matrix with

$$\begin{aligned} det V_{d+1} &= \begin{vmatrix} 1 & i_1 & i_1^2 & \cdots & i_1^d \\ 1 & i_2 & i_2^2 & \cdots & i_2^d \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & i_{d+1} & i_{d+1}^2 & \cdots & i_{d+1}^d \end{vmatrix} \\ &= \prod_{k>l} (i_k - i_l). \end{aligned}$$

The expression for $det\Lambda_{d+1}(\epsilon)$ is a polynomial in ϵ of degree d, which is not identically zero because of the non-vanishing highest order term.

The second matrix of interest has rows representing input objects $p_{i_1}, p_{i_2}, \ldots p_{i_d}$.

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

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

$$\begin{aligned} det\Delta_d(\epsilon) &= det\Delta_d \\ &+ (\epsilon^k \ terms, 1 \leq k \leq d-1) \\ &+ \epsilon^d (\prod_{k=1}^{d} i_k) \, det V_d, \end{aligned}$$

where V_d is a $d \times d$ Vandermonde matrix. Again, the highest order term is non-zero, hence the ϵ-polynomial is not identically zero.

Lemma 4.1 *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 algebraic branching program is non-singular and its determinant has constant sign.*

Proof From the theory of polynomials over ordered fields, any polynomial which is not identically zero, has a set of roots of codimension 1. Any algebraic set of roots, or zero set, whose dimension is at most 1, is either the entire (real) line or the union of a finite number of points.

The expressions for $det\Lambda_{d+1}(\epsilon)$ and $det\Delta_d(\epsilon)$ are polynomials in ϵ and are not identically zero. Thus, there exists a finite number of roots for each one. Letting ϵ_0 be the minimum positive such root, over all test determinants encountered, proves the lemma. □

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 the approach adopted by the two other techniques described above, [10] and [18]. Our perturbation scheme lends itself to a more efficient trick.

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

Having implicitly defined L, denoting by I_k the $k \times k$ unit matrix and relying on the fact that every Vandermonde matrix V_{d+1} is invertible, we have

$$
\begin{aligned}
&det\Lambda_{d+1}(\epsilon) = \\
&= \frac{1}{\epsilon} det(-V_{d+1})\, det((-V_{d+1}^{-1})L - \epsilon\, I_{d+1}) \\
&= \frac{1}{\epsilon}(-1)^{d+1}\, detV_{d+1}\, det(M - \epsilon\, I_{d+1}).
\end{aligned}
$$

Similarly, Vandermonde matrix V_d is used to express

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

where

$$
N = -\begin{bmatrix} i_1 & i_1^2 & \cdots & i_1^d \\ i_2 & i_2^2 & \cdots & i_2^d \\ \vdots & \vdots & & \vdots \\ i_{d+1} & i_{d+1}^2 & \cdots & i_{d+1}^d \end{bmatrix}^{-1} \Delta_d.
$$

Let s express the bit size of the input parameters and let $MM(k)$ denote the number of operations needed to multiply two $k \times k$ matrices.

Lemma 4.2 *Computing the sign of perturbed determinants* $\det\Lambda_{d+1}(\epsilon)$ *and* $\det\Delta_d(\epsilon)$ *under the algebraic branching program model can be done in* $\mathcal{O}(MM(d))$ *steps. These computations, under the bit model, have worst-case complexity* $\mathcal{O}(MM(d))\mathcal{O}(M(s) + M(d^2 \log n))$.

Proof Computations for both $det\Lambda_{d+1}(\epsilon)$ and $det\Delta_d(\epsilon)$ can be treated analogously, for they have the same two possible bottlenecks. Computing M or N takes $\mathcal{O}(MM(d))$ arithmetic operations. Computing $det(M - \epsilon\, I_{d+1})$ or $det(N - \epsilon\, I_d)$ respectively, is a characteristic polynomial computation for which there exists an algorithm by Keller-Gehrig [14] which requires $\mathcal{O}(MM(d))$ operations.

Clearly, the perturbation quantities have bit size $\mathcal{O}(d \log n)$. After inverting the Vandermonde matrix the resulting matrix has entries of bit size $\mathcal{O}(d^2 \log n)$. Then each operation has bit cost bounded by the maximum of $M(s)$ and $M(d^2 \log n)$. □

Proposition 4.3 *Consider algorithms that compute and branch on determinants of* Λ_δ *and* Δ_δ, *for* $\delta \leq d$, *where* d *is the dimension of the geometric space of the input objects. The perturbation defined by (1) is valid and does not change the running-time complexity of the algorithm under the algebraic branching program model. Under the bit model the complexity is increased by a factor* $\mathcal{O}(d^{2+\alpha})$, *where* α *is an arbitrarily small positive constant.*

Proof At branching nodes, the sign of the perturbed determinant is taken to be the sign of the lowest order non-vanishing term. If the original determinant is non-zero, it dominates the ϵ-polynomial. Otherwise, we simulate an artificial non-degenerate situation in which the input points are arbitrarily close to the original ones. Lemma 4.1 proves the consistency of our answers as ϵ approaches to zero on the positive real axis.

Lemma 4.2 asserts that the computation of a determinant takes $\mathcal{O}(MM(d))$ operations which is the original complexity of this operation since a $d \times d$ determinant had to be computed. Under the bit model, the original complexity per operation is $M(s) \geq M(\log n)$, since there are n distinct input objects. On perturbed input the worst-case bit complexity is $M(d^2 \log n)$ which is bounded by $\mathcal{O}(d^2 \log^2 d)M(\log n)$. Hence there is an extra factor $\mathcal{O}(d^2 \log^2 d) = \mathcal{O}(d^{2+\alpha})$, where α is a constant that can be arbitrarily small, as long as it remains positive. □

Even if d is not fixed, it is usually a small integer, independent of n. Then, the bit complexity is increased by a small factor and asymptotically remains the same.

4.2 Polynomial tests

We generalize our scheme into a randomized perturbation that applies to algebraic branching programs with arbitrary rational expressions as branching tests. Let d be the maximum total degree of any polynomial appearing in a vertex of some program. A branching vertex v decides on the sign of an arbitrary rational expression f, which we write as $f = p/q$. Both numerator and denominator are polynomials in the input variables $\mathbf{x}$ whose total degree obviously cannot exceed d.

If $\mathbf{x}$ belongs to $\mathbf{R}^n$, let $\mathbf{a} = (a_1, \ldots, a_n)$ be a particular input. By definition, this input is degenerate exactly when it causes a test expression, which is defined and not identically zero, to vanish. Here we are concerned only with such tests, i.e. tests f for which $p(\mathbf{x}) \not\equiv 0$ and $q(\mathbf{x}) \not\equiv 0$. For a given input $\mathbf{a}$, define the perturbed input $\mathbf{a}(\epsilon) = (a_1(\epsilon), \ldots, a_n(\epsilon))$ as follows:

$$a_i(\epsilon) = a_i + \epsilon r_i \tag{2}$$

where ϵ is an infinitesimal symbolic variable and r_i is a random integer. Computation on the perturbed input is carried out symbolically, since we never substitute a value for ϵ, which implies that results of computations and test expressions are rational expressions in ϵ. For branches, the sign of an ϵ-polynomial is taken to be the sign of its lowest-order non-zero term.

Each r_i is chosen uniformly over a range that depends on the desired probability of success. All claims in this section hold with probability at least $1 - 1/c$, for any positive real constant c. Then the bit size of the perturbation quantities is

$$\lg c + \lg d + (\lg 3)T + 1,$$

where lg expresses the logarithm of base 2 and T is the maximum number of branches on a computation path. Clearly, the total number of polynomials appearing at the numerator or denominator of a test expression is at most $2\,3^T$.

It is feasible that some set of random variables will not avoid degeneracies. In this case, the algorithm is restarted and the random variables are picked anew. We proceed immediately towards the central result of this section, which is that perturbation (2) has arbitrarily high probability to satisfy the validity condition.

Lemma 4.4 *Let the entries of $\mathbf{r} = (r_1, \ldots, r_n)$ be independently and uniformly chosen integers of $(\lg c + \lg d + (\lg 3)T + 1)$ bits each, for any positive c. Then, there exists with probability at least $1 - 1/c$, a positive real constant ϵ_0 such that, for every ϵ smaller than ϵ_0, every rational expression $f(\mathbf{a}+\epsilon\mathbf{r})$ 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 test 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 $2\,3^T d$.

Now we apply a lemma proven by Schwartz in [16]. For all i, r_i has $(\lg c + \lg d + (\lg 3)T + 1)$ bits, G is not the zero polynomial and has total degree $2\,3^T d$. Then the probability that $\mathbf{r}$, chosen uniformly at random, is a root of $G(\mathbf{a} + 1\mathbf{r})$ is at most $1/c$. All claims that follow hold with probability at least $1 - 1/c$.

First observe that none of the polynomials $g(\mathbf{a}+1\mathbf{r})$ vanishes at the particular $\mathbf{r}$, 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 codimension 1 which implies that the latter is either the entire (real) line or a finite point set. But $g(\mathbf{a} + \epsilon\mathbf{r}) \not\equiv 0$, therefore there is a finite number of real roots and we consider the minimum positive root for every g. Let ϵ_0 be the minimum over all polynomials g. □

What is the tradeoff in efficiency? The complexity of the algorithm is increased by the time required to manipulate the ϵ-polynomials symbolically. The degree of every polynomial in ϵ is the same as its degree in $\mathbf{x}$.

Lemma 4.5 *Perturbation (2) defines a new program that goes through $O(d)$ branching and $O(d \log^2 d)$ operation vertices for every branching and operation vertex respectively in the original program. The bit complexity of the new program depends on the bit size of the operands which is $O(ds + d \log d + dT)$, where s denotes the bit size of the input parameters and T the maximum number of branches on a computation path.*

Proof Instead of numeric computations we now have symbolic ones. Each operation $\circ \in \{+, -, \times, /\}$ involves multiplication of polynomials in one variable and a Greatest Common Divisor computation to reduce to lowest terms. The former takes time $O(d \log d)$ and the latter $O(d \log^2 d)$. Hence every operation now takes $O(d \log^2 d)$ numeric operations. Branching vertices must find the lowest non-vanishing term in the corresponding ϵ-expression, which takes $O(d)$ time. Formally, we have constructed a new program that contains, for every operation vertex of the original program, $O(d \log^2 d)$ unit-time vertices and, for every test of the original, $O(d)$ unit-time tests.

On perturbed input, arithmetic is performed on the original input parameters as well as on the perturbation quantities. They respectively start at bit size s and $\mathcal{O}(\log d + T)$, if c is fixed. They may give rise to $\mathcal{O}(ds)$-bit and $\mathcal{O}(d \log d + dT)$-bit quantities respectively, since d is the largest power of any polynomial. The largest of the three terms in $\mathcal{O}(ds + d \log d + dT)$ will determine the final bit complexity. □

Proposition 4.6 *The perturbation defined by (2) is valid with arbitrarily high probability. The running-time overhead to implement it under the algebraic branching program model is a $\mathcal{O}(d \log^2 d)$ factor, where d is the maximum degree in the input variables of any polynomial in the program. Under the bit model, the complexity is $\mathcal{O}(\phi^{3+\alpha}(n))$, where $\phi(n)$ denotes the original complexity and α is an arbitrarily small positive constant.*

Proof Test operations 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. For non-degenerate inputs, all query polynomials have a non-vanishing real part, i.e. a term independent of ϵ which dominates the sign. As far as the computation path is concerned, it is as if the perturbation had never taken place. For degenerate inputs, the algorithm behaves as if it was given non-degenerate input values arbitrarily close to the original ones, since ϵ can become vanishingly small and, by lemma 4.4, not affect the outcome of any branching decision.

By lemma 4.5, every vertex in the given program corresponds to at most $\mathcal{O}(d \log^2 d)$ vertices in the new one. This establishes the algebraic complexity overhead.

With regard to the bit model, we ignore again the effect of s and write the complexity of the original program ϕ as $\Omega(\overline{T} + Ad + B\,M(d))$, with $\overline{T} \leq T$ denoting the number of branches, A the number of additions and subtractions and B the number of multiplications and divisions on the computation path that corresponds to the worst-case complexity. We shall abuse notation by letting various occurrences of α represent different constants in interval $(0, 1)$. Then we expand $\phi^{3+\alpha}$ and keep just a few terms:

$$\begin{aligned} \phi^{3+\alpha} &= \phi^{1+\alpha}\phi^{2+\alpha} \\ &= \phi^{1+\alpha}\,\Omega(A^{2+\alpha}d^{2+\alpha} + B^{2+\alpha}d^{2+\alpha}). \quad (3) \end{aligned}$$

The worst case computation path in the modified program goes through $\mathcal{O}(\overline{T}d)$ unit-cost branching vertices, $\mathcal{O}(Ad \log^2 d)$ numeric $\{+, -\}$ operations and $\mathcal{O}(Bd \log^2 d)$ numeric $\{*, /\}$ operations. Therefore, the new complexity is bounded by

$$\begin{aligned} \mathcal{O}(\ &\overline{T}d + Ad \log^2 d(d \log d + dT) + \\ &+ Bd \log^2 d\, M(d \log d + dT)\ ). \end{aligned}$$

Clearly, $\overline{T} \leq T = \mathcal{O}(\phi)$; furthermore $M(b) = \mathcal{O}(b \log b \log\log b)$. Each term in the last expression can be matched with a term in (3) that is asymptotically larger, hence proving the claim. □

Corollary 4.7 *If the bit complexity of the algorithm on unperturbed input lies in P or EXP, then its complexity on perturbed input also lies in P or EXP respectively.*

Proof Immediate from the previous proposition. □

5 Conclusion

We have studied algorithms modeled as algebraic branching programs, with inputs from an infinite ordered field. This paper describes direct 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.

We defined a deterministic method for algorithms with determinant tests and a randomized one for arbitrary test expressions. They both incur extra complexity factors that are constant in several cases. Moreover, polynomial and exponential time algorithms always remain in the same complexity class while being enhanced with the power to execute on arbitrary inputs. Both methods are discerned by their conceptual elegance and are significantly faster than previous ones.

It is 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 the output vertices in the algebraic branching program model are associated with semi-algebraic sets defined by the test polynomials encountered on the path from the input vertex. We may wish to perturb these sets into general position.

References

[1] Blum L., 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.*, Vol. 21, No. 1, pp. 1-46, July 1989.

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

[3] Canny J., Some Algebraic and Geometric Computations in PSPACE, *Proc. 20th ACM STOC,* pp. 460-467, 1988.

[4] Canny J.F., *The Complexity of Robot Motion Planning,* MIT Press, Cambridge, Mass., 1988.

[5] Canny J.F., Computing Roadmaps of Semi-Algebraic Sets, to appear in *Proc. 9th AAECC,* New Orleans, Oct. 1991.

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

[7] Chvátal V., *Linear Programming,* W.H. Freeman & Co., New York/San Francisco, 1983.

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

[9] Edelsbrunner H., *Algorithms in Combinatorial Geometry,* Springer-Verlag, Heidelberg, 1987.

[10] Edelsbrunner H. and E.P. Mücke, Simulation of Simplicity: A Technique to Cope with Degenerate Cases in Geometric Algorithms, *ACM Trans. Graphics,* Vol. 9, No. 1, pp. 67-104, 1990.

[11] Edelsbrunner H. and R. Waupotitsch, Computing a ham-sandwich cut in two dimensions, *J. Symbolic Comput. 2,* pp. 171-178, 1986.

[12] Emiris I., An Efficient Approach to Removing Geometric Degeneracies, Master's thesis, Computer Science Division, UC Berkeley, May 1991.

[13] Gårding L. and T. Tambour, *Algebra for Computer Science,* Springer-Verlag, New York, 1988.

[14] Keller-Gehrig W., Fast Algorithms for the Characteristic Polynomial, *Theor. Comp. Sci.,* Vol. 36, pp. 309-317, 1985.

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

[16] Schwartz J.T., Fast Probabilistic Algorithms for Verification of Polynomial Identities, *Jour. ACM,* Vol. 27, No. 4, pp. 701-717, 1980.

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

[18] Yap C.-K., Symbolic treatment of geometric degeneracies, *Proc. 13th IFIP Conference on System Modeling and Optimization,* Tokyo, 1987.

[19] Yap C.-K., A geometric consistency theorem for a symbolic perturbation scheme, *Proc. 4th ACM Symp. on Comp. Geometry,* Urbana, 1988.