Lecture XI
DEGENERACY AND PERTURBATION

Computational geometry developed as a branch of theoretical algorithmics whose principal interest is in understanding the inherent complexity of computational problems. As such, the algorithms on paper are usually not meant to be literally implemented. When practitioners attempt to read these descriptions as instructions for an implementation, they begin to notice statements of such as the following:

> *"Our algorithm accepts an input set S of points. For simplicity, we assume that the points in S are distinct, no 2 points are co-vertical, no 3 points are collinear, no 4 points are co-circular, etc. The reader can easily extend our algorithm to handle inputs that fail these assumptions."*

The inputs that satisfy avoids the special situations are said to be **non-degenerate** otherwise **degenerate**. The problem is that implementors have little interest in extending the algorithm to such degenerate inputs. Even when there is a desire to do this, the number of special cases can be overwhelming and tricky to exhaustively enumerate. Roughly speaking, the number of special cases for handling degeneracies increases exponentially with the dimension d of the underlying geometry (e.g., $S \in R^d$).

In other words, there is a gap between theoretical algorithms and practical algorithms. In this lecture, we explore some general techniques for bridging this gap. Let us call an algorithm **generic** if it works correctly only for non-degenerate inputs; it is **general** if it works correctly for all inputs, degenerate or not. We specifically treat the gap between generic and general algorithms.

The theoretician may feel that the degenerate cases are rare. This is true in some measure-theoretic sense, but ignoring these cases will not do in an implementation. One impulse is to require the theoretician to supply details for the degenerate cases as well. Indeed, some has suggested this as a standard for publications in algorithms. This is[1] neither desirable nor enforceable. For instance, such details may not be the most instructive reading. As suggested above, converting a generic algorithm to a general algorithm by explicitly treating degeneracies can increase the size of the code be a large factor.

We propose the following framework: begin with a generic algorithm A for a problem P. We provide schemes for transforming A into a general algorithm A' for P. But what is A'? It cannot really be a full fledged algorithm for P since, by definition, it does not really handle degenerate cases. Yet as a general algorithm it does produce a reasonable output, even when the input is degenerate. We call this output a "perturbed output". This calls for another postprocessing algorithm B. This framework is illustrated in figure 1.

Demerits and Merits. It has been pointed out (see [1]) that (i) programming the degenerate cases may not be onerous, and (ii) perturbation can increase the run-time of specific instances by an exponential amount. A simple example is the convex hull of a set of n points in d dimensions. If all n points are coincident, the runtime is constant, but the convex hull of n points in general position is $\Theta(n^{\lfloor d/2 \rfloor})$. Users need to be aware of such costs.

Let us now argue in favor of this framework. By not constructing a general algorithm in the first place, aren't we doing more work in the end? We need to construct a generic A, transform it to A' and finally, provide a postprocessor B! We see in specific examples that this is not necessarily so. First of all,

[1] The algorithmic activity, like all human activity, is multifaceted and serves no single overriding concern, neither that of the implementors nor that of the theoreticians.

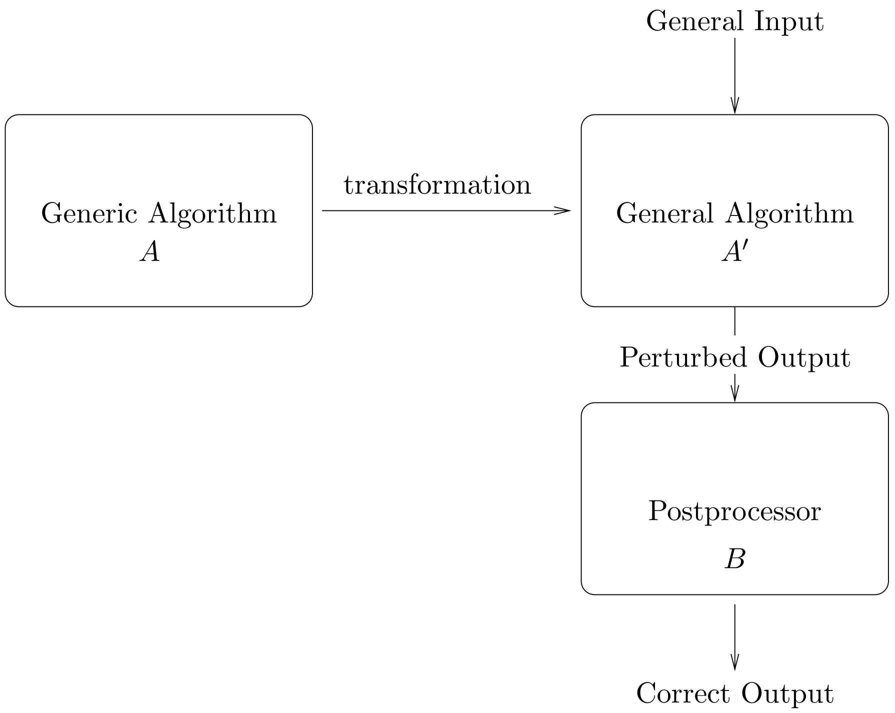

Figure 1: Generic to General Transformation

the transformation $A \mapsto A'$ can be quite simple, under certain conditions to be specified. Basically the idea is to perturbed any input so that it looks nondegenerate. Second, in view of the possible code bloat when we explicitly handle degeneracies, the work involved in constructing A' is negligible compared to a general algorithm. An example of this is the problem of constructing hyperplane arrangements. Finally, the postprocessing algorithm B in some examples turns out to be relatively simple. More importantly, B is not specific to A', but can be used with *any* transformed algorithm for P. For instance, if P is computing Voronoi diagrams of a point set, then B amounts to detecting Voronoi edges of length 0 and collapsing their end points. Thus any generic algorithm for P can take advantage of B.

An Example. It is instructive to work out the degenerate cases of some of the algorithms we have encountered so far in these lectures. These are not too difficult, but it can also serve to indicate the kind of tedium that our propose framework can eliminate.

Let us begin with the Bentley-Ottmann algorithm.

Exercises

Exercise 0.1: Work out the degenerate cases for the beneath-beyong algorithm for 3-D convex hull. ♢

End Exercises

§1. Model of Degeneracy

In the introduction, we described some typical degeneracy conditions on input points. What counts as degeneracy depends on the algorithm. Here are some more examples of degeneracy conditions, in case the input is a set of lines:

- a vertical line
- 2 parallel lines
- 2 perpendicular lines
- 3 concurrent lines
- 2 intersecting lines (when the lines are embedded in dimension greater that 2)
- the intersection points defined by pairs of lines satisfy the degeneracy conditions specified for point sets.

In case the input set is a set S of points and a set T of lines, we may count as degenerate if: (1) an input point lying on an input line, (2) an input line parallel to the line through 2 input points, etc.

Note that these degeneracy conditions can be specified as a polynomial $p(x_1, \ldots, x_m)$ vanishing on some of the input parameters.

We will call the above **á priori degeneracy conditions**. Usually, these á priori conditions are meant to prevent the occurrence of **á posteriori degeneracy conditions**. An input is said to be **ṕosteriori degenerate** for an algorithm if, in the course of executing the algorithm, some branching step takes a zero-branch. Here we assume the algorithm performs 3-way branching steps based on the sign of the evaluation of some real function

$$f(y_1, \ldots, y_m). \tag{1}$$

The sign of $f(y_1, \ldots, y_m)$ determines whether we take the negative-, zero- or positive-branch. So the zero-branch is considered degenerate.

We may compare á priori and ṕosteriori degeneracy with compile-time and run-time error conditions.

Our definition of degenerate inputs is **algorithm-induced**. We can say an input for a problem is **inherently degenerate** if it is degenerate for all algorithms for the problem. An example is an input set S of points for the convex hull problem. If three of the points in S are collinear and they lie on the boundary of the convex hull of S, we would say that S is inherently degenerate. But we focus on algorithm-induced degeneracy, as this is more important in practice.

We ought to say that the inputs are assumed to be exact whenever we discuss perturbation methods. When inputs are imprecise, we need to address several other issues first: what exactly do we mean by an inprecise input? What is an algorithm supposed to output? What does perturbation mean for imprecise inputs. These are beyond the present scope.

Exercises

Exercise 1.1: Provide polynomials whose vanishing corresponds to each to the degeneracy condition listed above. ♢

End Exercises

§2. Black Box Perturbation Scheme

In many algorithms, the test functions $f(y_1, \ldots, y_m)$ (1) used in branching tests are polynomials. Furthermore, the test parameters $y_1, \ldots, y_m$ are original input values. We say that the y_i's have depth 0 in this case. In general, the parameters y_i are computed from the input parameters, and we may define a suitable notion of "depth" for these computed values. In the following, assume

$$a_1, \ldots, a_n \tag{2}$$

are the numerical input parameters. There may be other input data which are non-numerical.

We will first describe a scheme described in [4] which works under three conditions:

- The test functions are polynomial.
- The test parameters have depth 0.
- The application is indifferent to arbitrarily small perturbations to the input parameters.

Later we note possibilities to relax some of these conditions. The idea is to construct a "black box" which, given any non-zero polynomial $p = p(x_1, \ldots, x_m)$ and any sequence of depth 0 parameters $\mathbf{b} = (b_1, \ldots, b_m)$, to output a non-zero sign

$$\mathtt{sign}_{\mathbf{b}}(p) \in \{-1, +1\}.$$

Moreover, if $p(\mathbf{b}) = p(b_1, \ldots, b_m)$ is non-zero, then $\mathtt{sign}_{\mathbf{b}}(p)$ is just the sign of $p(\mathbf{b})$. Thus, the interesting case happens when $p(\mathbf{b}) = 0$: in effect, we are computing a "signed zero", $+0$ or -0.

Using the Black Box. As in figure 1, we begin with a generic algorithm A. We construct the general algorithm A' by replacing each polynomial-test $p(\mathbf{b})$ in A with a call to this black box. The algorithm then continues with the positive or negative branch after the test, depending on $\mathtt{sign}_{\mathbf{b}}(p)$. Even if the input is degenerate, no zero-branch is ever taken.

Before describing our solution, consider the following possible approach. Randomly perturb the inputs $\mathbf{b}'$ and evaluate the polynomial $p(\mathbf{x})$ on the perturbed $\mathbf{b}'$. With high probability, $p(\mathbf{b}')$ is non-zero, as desired. But what if the result is zero? Perturb again. Unfortunately, we need to make sure that the new perturbation preserves the signs of all previously evaluated polynomials. This seems hard. Moreover, we must ensure that the perturbation $\mathbf{b}'$ is sufficiently small that it does not change the sign of any non-zero $p(\mathbf{b})$. We abandon this approach for now, although the idea of numerical perturbation can be made to work under suitable conditions (see Section 4 below).

Admissible Black Box Schemes. Our scheme is based on the concept of "admissible orderings" which originally arose in Gröbner bases theory.

Let $\mathbf{x} = (x_1, \ldots, x_n)$ be real variables (corresponding to the input parameters (2)). Let $\mathtt{PP} = \mathtt{PP}(x_1, \ldots, x_n)$ denote the set of all power products over $x_1, \ldots, x_n$. Thus a typical power product has the form

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

where the e_i are natural numbers. Note that $1 \in \mathtt{PP}$ corresponding to $e_1 = e_2 = \cdots = e_n = 0$. If $e = (e_1, \ldots, e_n)$ we also write $w = x^e$. The **degree** of x^e is $\sum_{i=1}^n e_i$, also denoted $|e|$. We say a total ordering $\le_A$ on $\mathtt{PP}$ is **admissible** if for all $u, v, w \in \mathtt{PP}$, (1) $1 \le_A w$, and (2) $u \le_A v$ implies $uw \le_A vw$.

There are infinitely many admissible orderings, but the two most familiar ones are **(pure) lexicographical ordering** $\le_{\mathtt{LEX}}$ and **total degree ordering** $\le_{\mathtt{TOT}}$. These are defined as follows: let $w = x^e$ and $v = x^d$ where $e = (e_1, \ldots, d_n)$ and $d = (d_1, \ldots, d_n)$.

- $w \le_{\mathtt{LEX}} v$ iff either $e = d$ or else $e_i < d_i$ where i is the first index where $e_i \ne d_i$.
- $w \le_{\mathtt{TOT}} v$ iff $|e| < |d|$ or else $w \le_{\mathtt{LEX}} v$.

Notice that both of these orderings depend on a choice of an ordering of the variables $x_1, \ldots, x_n$. So there are really $n!$ lexicographical (resp. total degree) orderings.

Now let $p(\mathbf{x}) \subseteq R[\mathbf{x}]$ be a real polynomial. For $w \in \mathtt{PP}$, we let $\partial_w(p)$ denote the partial derivative of $p(\mathbf{x})$ with respect to w. More precisely, if $w = x^e$ and $e = (e_1, \ldots, e_n)$ then

$$\partial_w(p) := \frac{\partial^{|e|} p}{\partial^{e_1} x_1 \partial^{e_2} x_2 \cdots \partial^{e_n} x_n}.$$

For instance, $w = x^3yz^3$ (we typically let $x = x_1, y = x_2, z = x_3$) then $\partial_w(p) := \frac{\partial^6 p}{(\partial x)^3 (\partial y)(\partial z)^2}$. If $p(x, y, z) = 2x^2z^3 - 7xy + y^5 + 10$ then $\partial_x(p) = 4xz^3 - 7y$, $\partial_{xy}(p) = 7$, $\partial_{z^2}(p) = 24xz$, $\partial_{xyz}(p) = \partial_{yz^2}(p) = 0$.

Fix any admissible ordering $\le_A$. We describe a blackbox sign function corresponding to $\le_A$. The ordering $\le_A$ gives an enumeration

$$(w_0, w_1, w_2, \ldots), \qquad w_i \le_A w_{i+1}$$

of all power products in $\mathtt{PP}$. Clearly, $w_0 = 1$. Define the sequence of polynomials

$$\partial(p) := (\partial_{w_0}(p), \partial_{w_1}(p), \cdots).$$

If $\mathbf{a} = (a_1, \ldots, a_n)$ then define

$$\partial(p; \mathbf{a}) := (\partial_{w_0}(p)(\mathbf{a}), \partial_{w_1}(p)(\mathbf{a}), \cdots).$$

So $\partial(p; \mathbf{a})$ is an infinite sequence of numbers obtained by evaluating each polynomial of $\partial(p)$ at $\mathbf{a}$. We finally define our "black box sign" $\mathtt{sign}_{\mathbf{a}}(p)$ to be the sign of the first non-zero value in $\partial(p; \mathbf{a})$. Note that if $p \ne 0$ then $\partial_{w_i}(p)$ is a non-zero constant for some a. Hence $\mathtt{sign}_{\mathbf{a}}(p)$ is well-defined for all non-zero p. By definition, $\mathtt{sign}_{\mathbf{a}}(p) = 0$ when p is the zero polynomial. Also write

$$\mathtt{sign}_{\mathbf{a}, \le_A}(p) \tag{3}$$

for $\mathtt{sign}_{\mathbf{a}}(p)$ to indicate the dependence on $\le_A$. We will call $\mathtt{sign}_{\le_A, \mathbf{a}}$ an **admissible sign function**.

Example: SoS Scheme of Edelsbrunner and Mücke. Suppose H is a set of n lines in the plane. Each $h_i \in H$ $(i = 1, \ldots, n)$ has the equation $y = a_i x + b_i$. Thus the input parameters are $(a_1, b_1, a_2, b_2, \ldots, b_n)$. The problem is to compute the arrangement of cells defined by these lines: each cells has dimensions 0, 1 or 2 where 0-cells are just points of intersection defined by the lines, 1-cells are open line segments bounded by 0-cells, and 2-cells are the open connected polygonal regions bounded by the lines. The algorithm for computing an arrangement is fairly easy in the generic case. But dealing with degeneracies is complicated, and it gets much worse in higher dimensions. Hence there is strong motivation to use a generic algorithm, and to perturb the input to ensure non-degeneracy. Suppose 3 distinct lines h_i, h_j, h_k are concurrent (passes through a common point). This is a degeneracy. It amounts to the vanishing of the following determinant:

$$\delta_{ijk} = \det \begin{bmatrix} a_i & b_i & 1 \\ a_j & b_j & 1 \\ a_k & b_k & 1 \end{bmatrix}. \tag{4}$$

We say H is **simple** if no 3 distinct lines are concurrent. We want to perturb the input parameters so that H is simple. For this purpose, let $\varepsilon > 0$ be a small value (to be clarified) and define $h_i(\varepsilon)$ with equation

$$\begin{aligned} y &= a_i(\varepsilon)x + b_i(\varepsilon), \\ a_i(\varepsilon) &= a_i + \varepsilon^{2^{2i}}, \\ b_i(\varepsilon) &= b_i + \varepsilon^{2^{2i-1}}. \end{aligned}$$

The corresponding determinant, expanded as a polynomial in ε is given by

$$\begin{aligned} \delta_{ijk}(\varepsilon) &= \det \begin{bmatrix} a_i(\varepsilon) & b_i(\varepsilon) & 1 \\ a_j(\varepsilon) & b_j(\varepsilon) & 1 \\ a_k(\varepsilon) & b_k(\varepsilon) & 1 \end{bmatrix} \\ &= \det \begin{bmatrix} a_i & b_i & 1 \\ a_j & b_j & 1 \\ a_k & b_k & 1 \end{bmatrix} \\ &\quad -\varepsilon^{2^{2i-1}} \det \begin{bmatrix} a_j & 1 \\ a_k & 1 \end{bmatrix} \\ &\quad +\varepsilon^{2^{2i}} \det \begin{bmatrix} b_j & 1 \\ b_k & 1 \end{bmatrix} \\ &\quad +\varepsilon^{2^{2j-1}} \det \begin{bmatrix} a_i & 1 \\ a_k & 1 \end{bmatrix} \\ &\quad +\varepsilon^{2^{2j-1}+2^{2k-1}} \\ &\quad +\cdots. \end{aligned}$$

The key observation is that each coefficient of this polynomial is a subdeterminant of Δ. Moreover, if ε is sufficiently small, and $i < j < k$, then the sign of $\delta_{ijk}(\varepsilon)$ is obtained by evaluating these coefficients in the order shown above (in order of increasing power of ε). The first term is therefore Δ. We stop at the first non-zero coefficient, as this is the sign of the overall polynomial.

This ε-scheme is called **Simulation of Simplicity** (SoS) and clearly works in any dimension. We leave it as an exercise to show that SoS amounts to an admissible sign function, based on a suitable lexicographic ordering $\leq_{\mathtt{LEX}}$.

We next address three key issues:

- The $\mathtt{sign}(p(\mathbf{x}), \mathbf{b})$ must satisfy some notion of consistency. What is it?
- How should we interprete the output from the generic algorithm when the input is degenerate?
- How can we implement such a black box?

Exercises

Exercise 2.1: Show that the sign given to $\Delta_{ijk}(\varepsilon)$ is exactly the sign given by a suitable admissible sign function. In fact, the admissible ordering is $\le_{\mathtt{LEX}}$ for a suitable ordering of the input parameters. $\diamondsuit$

End Exercises

§3. Algebraic Consistency

We address the consistency of admissible sign functions. We will use an algebraic approach here.

Let R be any commutative ring with unit 1. For instance, $R = Z$ or $R = R$. A function $\sigma : R \to \{-1, 0, +1\}$ is called a **sign function** for R provided it satisfies these 4 axioms. For all $a, b \in R$:

(A0) $\sigma(a) = 0$ iff $a = 0$.

(A1) $\sigma(-a) = -\sigma(a)$.

(A2) $\sigma(a) > 0$ and $\sigma(b) > 0$ implies $\sigma(a + b) > 0$.

(A3) $\sigma(ab) = \sigma(a)\sigma(b)$.

If σ is a sign function for R, then it induces a total ordering $<_\sigma$ on R where

$$a <_\sigma b \Rightarrow \sigma(a - b) = -1.$$

It is easy to verify that $<_\sigma$ is indeed a total ordering: if a, b are distinct elements in R, then $a <_\sigma b$ or $b <_\sigma a$, by (A0). Moreover, (A1) says that only one of $a <_\sigma b$ or $b <_\sigma a$ must hold. Transitivity follows from (A2).

We need the following lemma about admissible orderings $\le_A$:

Lemma 1 *Let $u, v, u', v' \in PP$ and either $u \ne u'$ or $v \ne v'$. Then $uv \le_A u'v'$ implies $u <_A u'$ or $v <_A v'$.*

See [4] for a proof. The next theorem constitute our "algebraic consistency" claim.

Theorem 2 *Let R be any ordered ring and $S = R[x_1, \ldots, x_n]$. Fix $\mathbf{a} = (a_1, \ldots, a_n) \in R^n$ and any admissible ordering $\le_A$. Let $\sigma = \sigma_{\mathbf{a}, \le_A}$ be the function that assigns to each polynomial $p \in S$ the sign $\mathtt{sign}_{\le_A}(p; \mathbf{a})$ (see (3)). Then σ is a sign function for S.*

Proof. Recall that for a non-zero $p \in S$, our black box sign, $\sigma(p)$, is the first non-zero value in the sequence $\partial(p; \mathbf{a})$. Axiom (A0) is immediate. Axiom (A1) comes from the fact that $\partial(-p; \mathbf{a})$ is obtained from $\partial(p; \mathbf{a})$ be negating every value. Axiom (A2) comes from the fact that $\partial(p+q; \mathbf{a}) = \partial(p; \mathbf{a}) + \partial(q; \mathbf{a})$ (componentwise addition). To show (A3), let the first non-zero entry of $\partial(p; \mathbf{a})$ be $\partial_u(p)(\mathbf{a})$ and the first non-zero entry of $\partial(q; \mathbf{a})$ be $\partial_v(q)(\mathbf{a})$, for some $u, v \in \mathtt{PP}$. Then

$$\partial_{uv}(pq) = \sum_{u'} \partial_{u'}(p)\partial_{v'}(q)$$

where $u' \in \mathtt{PP}$ ranges over all divisors of uv and $v' = uv/u'$. But the previous lemma, unless $u = u'$ and $v = v'$, $u'v' \leq_A uv$ implies $u >_A u'$ or $v >_A v'$. Hence $\partial_{u'}(p)(\mathbf{a}) = 0$ or $\partial_{v'}(p)(\mathbf{a}) = 0$. This proves

$$\partial_{uv}(pq)(\mathbf{a}) = \partial_u(p)(\mathbf{a})\partial_v(q)(\mathbf{a}).$$

The same argument shows that for all $w <_A uv$,

$$\partial_w(pq)(\mathbf{a}) = 0.$$

Thus, the first non-zero entry of $\partial(pq; \mathbf{a})$ has the sign $\partial_u(p)(\mathbf{a})\partial_v(q)(\mathbf{a}) = \sigma(p)\sigma(q)$, as desired. **Q.E.D.**

Example. The following problem arises in the computation of Euclidean maximum spanning tree [2]. Given n points $p_1, \ldots, p_n$ where $p_i = (x_i, y_i)$, we want to totally sort the set

$$\{D_{ij} : i, j = 1, \ldots, n, (i \neq j)\}$$

where $D_{ij} = \|p_i - p_j\|^2 = (x_i - x_j)^2 + (y_i - y_j)^2$. When $D_{ij} = D_{k\ell}$, Monma et al [2] break ties using the following rule: let $i_1 = \min\{i, j, k, \ell\}$ and relabel $\{i, j, k, \ell\} \setminus \{i_1\}$ as $\{j_1, i_2, j_2\}$ such that

$$\{\{i_1, j_1\}, \{i_2, j_2\}\} = \{\{i, j\}, \{k, \ell\}\}.$$

There are two cases:
(A) If $i_1 \notin \{i, j\} \cap \{k, \ell\}$ then make $D_{i_1,j_1} > D_{i_2,j_2}$ iff $x_{i_1} \geq x_{j_1}$.
(B) If $\{i_1\} = \{i, j\} \cap \{k, \ell\}$ then, without loss of generality, assume $i_1 = i_2$. Make $D_{i_1,j_1} > D_{i_2,j_2}$ iff $2x_{i_1} \geq x_{j_1} + x_{j_2}$.

Let us define the function σ which assigns to polynomial of the form $p = D_{ij} - D_{k\ell}$ a sign $\sigma(p) \in \{-1, +1\}$. The above scheme amounts to defining $\sigma(D_{ij} - D_{k\ell}) = +1$ iff $D_{ij} > D_{k\ell}$ (breaking ties as needed). An exercise below shows that σ arises from an admissible sign function.

Geometric Consistency. To motivate the need for a geometric notion of consistency, suppose that we have n collinear points, $q_1, \ldots, q_n \in R^2$. Let Δ_{ijk} denote determinant whose sign is the $LeftTurn(q_i, q_j, q_k)$ predicate (Lecture II.1). Since the lines are collinear, $\Delta_{ijk} = 0$ for all i, j, k. On the other hand, our black box will assign a non-zero sign $s_{ijk} \in \{-1, +1\}$ to Δ_{ijk}. The geometric consistency question asks to know if there is an actual perturbation of $q_1, \ldots, q_n$ which achieves these s_{ijk}'s. This question is answered in the affirmative in [3].

It is a consequence of the next theorem. For $\delta > 0$ and $\mathbf{a}, \mathbf{a}' \in R^n$, we say $\mathbf{a}'$ is a **δ-perturbation** of $\mathbf{a}$ if $\mathbf{a} = (a_1, \ldots, a_n)$ and $\mathbf{a}' = (a'_1, \ldots, a'_n)$ and $|a'_i - a_i| \leq \delta$ for all $i = 1, \ldots, n$.

Theorem 3 *Let σ be any black box sign function for $R[x_1, \ldots, x_n]$ (based on some admissible ordering and a choice of $\mathbf{a} \in R^n$). For any finite set $P \subseteq R[x_1, \ldots, x_n]$ of polynomials and any $\delta > 0$, there exists $\mathbf{a}' = (a'_1, a'_2, \ldots, a'_n)$, a δ-perturbation of $\mathbf{a}$, such that for all $p \in P$,*

$$\sigma(p) = \mathtt{sign}(p(\mathbf{a}')).$$

In other words, the signs assigned by our black box to any finite set P of real polynomials can be achieved by an actual perturbation. Suppose A' is the transformed algorithm in figure 1, and $\mathbf{a}$ is any input for A'. Consider the branching (computation) path π through A' that is taken by input $\mathbf{a}$. This result shows that there is an arbitrarily small perturbation $\mathbf{a}'$ of $\mathbf{a}$ such that $\mathbf{a}'$ is non-degenerate for A' and $\mathbf{a}'$ would exact the same path π.

Exercise 3.1: Show the following properties of $<_\sigma$ where σ is a sign function on R. For all $a, b, c, d \in R$:
(a) $a <_\sigma b$ implies $a + c <_\sigma b + c$.
(b) $a <_\sigma b$, $\sigma(c) > 0$ implies $ac <_\sigma bc$.
(c) If b^{-1}, a^{-1} exists, then $a <_\sigma b$ iff $b^{-1} <_\sigma a^{-1}$.
(d) $a + b <_\sigma c + d$ implies $a <_\sigma c$ or $b <_\sigma d$.
(e) $\sigma(a) = \sigma(b) = +1$ and $ab <_\sigma cd$ implies $a <_\sigma c$ or $b <_\sigma d$. ♢

Exercise 3.2: Show that the function σ defined by the Monma, et al, scheme arises from an admissible sign function. ♢

End Exercises

§4. Numerical Perturbation

We present a numerical method for perturbing determinants, from Canny and Emiris.

§5. Perturbing towards Non-Degenerate Instances

A fundamentally different idea for perturbation is proposed by Raimund Seidel: for any problem P and for each input size n, suppose we can find a non-degenerate instance G_n. Then for any input instance I of size n, we want to use the perturbed instance

$$I + \varepsilon G_n$$

where $\varepsilon > 0$ is a small constant.

Recall that one of our basic goals figure 1 is to transform a generic A to a general A'. In this case, we need to be able to generate G_n on the fly. This turns out to be easy in some problems. Consider the problem P of computing the convex hull of points in R^d. For each n, there are well-known constructions for a set G_n of n points in general position. Basically, G_n can be any n distinct points chosen on the moment curve,

$$C = \{(t, t^2, \ldots, t^d) : t \in R\}$$

References

[1] C. Burnikel, K. Mehlhorn, and S. Schirra. On degeneracy in geometric computations. *Proc. 5th ACM-SIAM Symp. on Discrete Algorithms*, pages 16–23, 1995.

[2] C. Monma, M. Paterson, S. Suri, and F. Yao. Computing Euclidean maximum spanning trees. *Algorithmica*, 5:407–419, 1990. Also: 4th ACM Symp.Comp.Geom. (1988).

[3] C. K. Yap. A geometric consistency theorem for a symbolic perturbation scheme. *J. of Computer and System Sciences*, 40(1):2–18, 1990.

[4] C. K. Yap. Symbolic treatment of geometric degeneracies. *J. of Symbolic Computation*, 10:349–370, 1990. Also, *Proc. International IFIPS Conf. on System Modelling and Optimization,* Tokyo, 1987, Lecture Notes in Control and Information Science, Vol.113, pp.348-358.