J. Symbolic Computation (1990) **10**, 281–309

An Algorithm for Generalized Point Location and Its Applications

BERNARD CHAZELLE[1] AND MICHA SHARIR[2]

(1) Dept. of Computer Science, Princeton University, Princeton, NJ 08544, U.S.A.
(2) School of Mathematical Sciences, Tel Aviv University, Tel Aviv, Israel.

(*Received* 20 *May* 1988)

We show that Collins' classical quantifier elimination procedure contains most of the ingredients for an efficient point location algorithm in higher-dimensional space. This leads to a polynomial-size data structure that allows us to locate, in logarithmic time, a point among a collection of real algebraic varieties of constant maximum degree, assuming that the dimension of the ambient space is fixed. This result has theoretical bearings on a number of optimization problems posed in the literature. It also gives a method for solving multidimensional searching problems in polynomial space and logarithmic query time.

1. Introduction

The central theme of multidimensional searching is the organization of a database to which queries of a chosen type can be made. The term *locus approach* refers to the particular strategy that regards a query as a point in higher-dimensional space: the idea is to subdivide the query space into equivalence classes and thus reduce query-answering to point location. This approach was followed by Dobkin & Lipton (1976), who devised an efficient searching algorithm for hyperplanes in E^d, for any fixed d. Yao & Yao (1985) have observed that the constraint manifold can usually be made linear by throwing in additional variables, if needed. Considering (as will be shown below) that the

0747-7171/90/090281+29 $03.00/0

preprocessing is doubly exponential in the number of variables, however, the linearization method may not always be so desirable. We will show here that Dobkin and Lipton's method can be generalized directly to handle real arbitrary algebraic varieties. The generalization is based on Collins' cylindrical algebraic decomposition (Collins, 1975, Arnon et al., 1984a,b). The main idea is to transform Collins' algorithm into a data structure and add various bells and whistles to support fast searching.

For a precise statement of our results we need to introduce a few notions. Let $\mathcal{F} = \{P_1, \ldots, P_n\}$ be a set of n d-variate polynomials with rational coefficients and norm-length at most ℓ. (The *norm-length* of a polynomial with integral coefficients is the number of bits needed to represent the sum of the absolute values of its coefficients. If the coefficients are rational, the norm-length is that of the integral polynomial obtained by putting all coefficients over a common denominator.) We also assume that the maximum degree of these polynomials is bounded above by a constant. Note that to allow all polynomials to be distinct, it is necessary to let ℓ be at least on the order of $\log n$. In practice one should expect ℓ to be at most polylogarithmic in n.

The *generalized point location problem* concerns the fast evaluation of the predicate

$$\left[\exists i\ (1 \leq i \leq n) \,|\, P_i(x) = 0\right],$$

for any query point $x \in E^d$. A simple true/false answer being a little too terse, we require supplementary information. If the predicate is true then some witness i such that $P_i(x) = 0$ should be provided (note that requiring the reporting of all such indices might by itself preclude a fast response). If, on the other hand, the predicate is false, then x lies in a connected region of the open set $\mathcal{C} = \left\{ x \in E^d \mid \prod_{1 \leq i \leq n} P_i(x) \neq 0 \right\}$. In that case, the desired output is usually the value $f(x)$ of some function f which is invariant over each connected region of $\mathcal{C}$. The preprocessing will compute a distinct algebraic point, a *sample*, in each region of $\mathcal{C}$ and evaluate f at the sample points.

The problem is a direct generalization of the well-known *planar point location* problem. Previous work on point location with nonlinear boundaries has been limited to the case $d = 2$, culminating in the optimal algorithms of Cole (1986) and Edelsbrunner et al. (1986) for subdivisions with "monotone" curves, and that of Sarnak and Tarjan (1986) for more general subdivisions. As will be shown below, a Collins decomposition

provides a simple framework for solving the point location problem in full generality. We will describe a data structure of size $O(n^{2^d-1})$ which allows us to answer any query in $O(\log n)$ time; the time needed to build the data structure is $O\left(n^{2^{d+6}}\right)$.

These bounds hold in the traditional unit-cost *RAM* model (Aho et al. 1974). One will notice, for example, that the norm-length ℓ does not even appear in the bounds. The reason is that in the unit-cost model any integer operation takes constant time, regardless of the length of the integers involved. Of course, this may sometimes hide the true cost of a computation if the integers become very large. Traditionally, algorithms in computational geometry have tended to ignore the true cost of precise rational arithmetic, although this issue has recently started to gain importance (see Dobkin and Silver, 1988, Hoffmann et al., 1988). In the case of our data structure, however, this cost must be taken into account. The algorithm involves iterated computations of polynomial greatest common divisors, subresultants, Sturm sequences, etc., all of which tend to inflate the size of the coefficients of the polynomials.

The complexity analysis of our data structure follows (Collins, 1975) without the added burden of storing cylindrical algebraic samples (though such samples might be computed, used in preprocessing, and thrown away). If we start out with d-variate polynomials of constant degree and norm-length ℓ, the preprocessing will only generate k-variate polynomials ($k \leq d$) of constant maximum degree and norm-length $O(\ell)$. Moreover, as follows from (Collins, 1975), all operations on the coefficients of the polynomials can be carried out in a number of bit operations at most cubic in their norm-length. This means that in order to obtain upper bounds on the bit complexity of the algorithm it suffices to multiply the unit-cost bounds given above by ℓ^3. This would give us a query time of $O(\ell^3 \log n)$ and a preprocessing time of $O\left(\ell^3 n^{2^{d+6}}\right)$. As long as a computer word can store up to ℓ bits the storage requirement is (asymptotically) the same in both models of computation. Although our underlying assumption will be the unit-cost model, we will also mention the bit complexity of an algorithm whenever there is a discrepancy between the two models. A final word concerning the dependency of the algorithm on the degree of the polynomials. We caution that our algorithm, like Collins', produces auxiliary polynomials whose maximum degrees can be truly enormous. This can add a large multiplicative factor, albeit constant, to the complexity of the algorithms: say, around b^{4^d}, where b the maximum degree of the input polynomials. We will ignore this

dependency in the subsequent analysis.

Interestingly, our data structure matches Dobkin and Lipton's (1976) in terms of storage requirements, both being $O(n^{2^d-1})$. Although the size of our data structure is polynomial in n, the magnitude of the exponent puts a severe limitation on its practicality. From a theoretical standpoint, however, this result has direct application to multidimensional searching: we will discuss this relationship in some detail later. The algorithm also has somewhat unexpected ramifications. We will use it as a tool in the solution of several problems. Two of them, posed by McKenna (1986), seek (i) the longest line segment fully contained within a given n-gon and (ii) the minimum "vertical" distance between two collections of red and blue segments in 3-space (see section 5 for details). Another one, due to Atallah (1985), asks at which time the convex hull of n points moving in the plane will first enter its final, steady configuration. Using our point location algorithm, combined with a batching technique originally proposed by Yao (1982), we are able to solve these problems, as well as other related ones, in subquadratic (albeit ever so close to quadratic) time.

These are a few concrete exemplifications of a more general principle, which is one of the main consequences of this paper. Many optimization problems in computational geometry have trivial quadratic solutions. Typically these problems involve two sets A, B of n objects each, and ask for the pair $(a, b) \in A \times B$ that minimizes some cost function, or satisfies some predicate; think, for example, of the diameter or closest-pair problem (Preparata and Shamos, 1985). A considerable amount of recent work in computational geometry can be regarded as attempts to beat this trivial quadratic bound by building clever data structures which reduce the number of pairs $a \in A, b \in B$ that need testing (e.g., the Voronoi diagram for the closest-pair problem). Our results imply that if the interaction of a single pair $a \in A, b \in B$ can be stated as an algebraic expression (possibly involving Boolean algebraic predicates) in the real parameters specifying a, b, then the data structure that we develop can be used to reduce the problem complexity to subquadratic.

In section 2 we review the algebraic backdrop behind the algorithms, and in section 3 we describe the point location data structure in detail. Section 4 discusses the relevance of the algorithm to multidimensional searching in general. In section 5 we tackle McKenna's problems by reducing them to a more general optimization question of the sort just

mentioned. We attack Atallah's problem in section 6, and give a more general discussion of the underlying technique in section 7.

2. The Algebraic Machinery

Most of the algebraic notions involved in this work can be found exposed in great detail in (Collins, 1975) and (Schwartz and Sharir, 1983). We have tried to adhere to the terminology used in these papers as much as possible. The fundamental algebraic concepts can be found in van der Waerden's classic text (van der Waerden, 1953), while for the specialized treatment of resultants and subresultants used in the paper the reader should turn to (Brown and Traub, 1971).

I) *Collins' Decidability Theorem*: In 1948 Tarski proved that every statement in elementary algebra (which is, the elementary theory of real-closed fields) is decidable (Tarski, 1948). The non-elementary procedure given by Tarski was subsequently improved (computationally) in a number of different ways by several researchers (e.g., Seidenberg, 1954, Cohen, 1969, Collins, 1975, Monk and Solovay, 1974, Ben-Or et al., 1984). For the purpose of the present work, we shall use Collins' decision procedure as a guiding framework. Let a *standard prenex formula* be any logical sentence of the form

$$(Q_k x_k)(Q_{k+1} x_{k+1}) \dots (Q_d x_d)\, \phi(x_1, \dots, x_d),$$

where each Q_i is a universal or existential quantifier and $\phi(x_1, \dots, x_d)$ is a quantifier-free formula made of Boolean connectives, standard comparators, and polynomials with rational coefficients in the real variables $x_1, \dots, x_d$. A logical sentence is called an *atomic formula* if it is free of quantifiers and logical connectives.

Theorem 1. (Collins, 1975) – *Let Φ be an arbitrary standard prenex formula with d variables, c atomic formulas, m polynomials of degree at most b in any single variable, with all integral coefficients of length at most ℓ. Whether Φ is true or false can be decided in at most $c\ell^3(2b)^{2^{2d+8}} m^{2^{d+6}}$ bit operations.*

II) *The Cylindrical Algebraic Decomposition*: This section reviews the essential components of Collins' decomposition needed for the point location algorithm. We include this discussion to make our exposition self-contained. The reader fully familiar with Collins' work may skip the next paragraphs.

A *Collins decomposition* of the d-dimensional Euclidean space E^d is a refinement of the decomposition of E^d induced by a finite collection of real algebraic varieties. (Each polynomial defining a variety is sign-invariant over each region of the decomposition.) The key concept is that of a *cylindrical algebraic decomposition* (or *cad*, for short). A d-dimensional *cad* is a partitioning of E^d defined inductively as follows.

(i) For $d = 1$, a *cad* is a finite set of disjoint open intervals and singletons whose union forms E^1. Each singleton contains an algebraic number: see a discussion later in this section on how to store algebraic numbers.

(ii) For $d > 1$, a *cad* K is defined in terms of a *cad* K' of E^{d-1} and a d-variate polynomial $P(x_1, \ldots, x_{d-1}, y)$ with rational coefficients. Let $K' = \{c_1, \ldots, c_\mu\}$; for each $c_i \in K'$ there exists an integer ν_i such that for each $x = (x_1, \ldots, x_{d-1}) \in c_i$, $P(x, y)$, regarded as a polynomial in y, has ν_i real roots $f_{i,1}(x) < \cdots < f_{i,\nu_i}(x)$, each of which is a continuous function in x over c_i. If $\nu_i = 0$, set $c_{i,1} = c_i \times E^1$. If $\nu_i > 0$, set $c_{i,2j} = \{ (x, f_{i,j}(x)) \mid x \in c_i \}$ for $1 \leq j \leq \nu_i$, and set $c_{i,2j+1} = \{ (x, y) \mid x \in c_i$ and $f_{i,j}(x) < y < f_{i,j+1}(x) \}$ for $1 \leq j < \nu_i$. Also, put $c_{i,1} = \{ (x, y) \mid x \in c_i$ and $y < f_{i,1}(x) \}$ and $c_{i,2\nu_i+1} = \{ (x, y) \mid x \in c_i$ and $f_{i,\nu_i}(x) < y \}$. Finally K is defined as the set of cells $\{c_{1,1}, \ldots, c_{1,2\nu_1+1}, \ldots, c_{\mu,1}, \ldots, c_{\mu,2\nu_\mu+1}\}$.

Following (Schwartz and Sharir, 1983) we call P the *base polynomial* of the *cad*. Informally, the cells of K can be formed by considering the cylinders based at each $c \in K'$ and chopping them off with the real hypersurface $P(x_1, \ldots, x_d) = 0$. Since K is defined in terms of a unique *cad* of lesser dimension, by induction, it defines an *induced cad* for each E^k $(1 \leq k < d)$. Incidentally, one should note that each cell of K is "well-behaved," in the sense that it is topologically equivalent to a relatively open ball of dimension at most d.

For our purposes the base polynomial P will always be of the form $\prod_{1 \leq i \leq n} P_i$, where $\mathcal{F} = \{P_1, \ldots, P_n\}$ is a collection of d-variate polynomials with rational coefficients. The key feature of a *cad* is that for each $c \in K$ and each $P_i \in \mathcal{F}$, the value of $P_i(x)$ is either zero over the entire cell c, or it keeps the same sign over the cell: a *cad* which satisfies this property is said to be $\mathcal{F}$-invariant. Besides introducing the concept itself, the main contribution of (Collins, 1975) was to prove that any collection $\mathcal{F}$ admits of an $\mathcal{F}$-invariant *cad* and that it can be constructed fairly efficiently (all things considered).

To simplify the computations (as well as carry the analysis further to determine the adjacencies between the cells of a *cad*) Schwartz and Sharir introduce the useful concept of a *well-based* decomposition: K is said to be well-based if, when regarded as a univariate polynomial in y, the base polynomial $P(x,y)$ is not identically zero for any given value of x in E^{d-1} (Schwartz and Sharir, 1983). They show that in that case each root function $f_{i,j}$ (defined over $c_i \in K'$) can be extended continuously over the closure of c_i. Informally, this means that every line $(x_1,\ldots,x_{d-1}) \times E^1$ intersects the algebraic variety $P(x,y) = 0$ only in a finite number of points. These intersections will form the basis of the binary search underlying the point location algorithm to be presented in the next section. How can we ensure that a decomposition is well-based? Since point location is defined independently of a coordinate system, we can always modify the frame of reference to ensure this condition. As a matter of fact, it is suggested in (Schwartz and Sharir, 1983) that a few random perturbations of the original coordinate system might be the best strategy in practice. (They also give a method for checking if a *cad* is well-based.) For our purposes, a well-based decomposition is convenient but not necessary. Therefore another solution is, of course, not to worry about it and simply ensure that the search procedure is robust enough to handle this type of degeneracy.

Following Collins' terminology, we define an *algebraic sample* of K as a set of points with algebraic coordinates, one in each cell of K (recall that a real number is algebraic if it is a root of a polynomial with integer coefficients). An algebraic sample is *cylindrical* (abbreviated *cas*) if either $d = 1$ or the set of $d-1$ first coordinates of each point forms a *cas* of K'. If $\{c_{i,1},\ldots,c_{i,2\nu_i+1}\}$ is the set of cells of K associated with the cell c_i of K', the sample points in each $c_{i,j}$ $(1 \le j \le 2\nu_i+1)$ all share the same first $d-1$ coordinates.

III) *The Collins Construction*: We begin with a short review of Collins' algorithm. Let Q be a d-variate polynomial of degree p with real rational coefficients. We can write $Q(x_1,\ldots,x_d)$ as a polynomial $\sum_{0\le i\le p} Q_i(x_1,\ldots,x_{d-1})x_d^i$ of a single variable x_d, with coefficients in the ring of $(d-1)$-variate polynomials with rational coefficients. Let $\deg(Q) = p$ be the degree of Q in x_d and let $\mathrm{ldcf}(Q) = Q_p(x_1,\ldots,x_{d-1})$ denote the leading nonzero coefficient of Q. We define the *reductum* of Q, denoted $\mathrm{red}(Q)$, as the polynomial $\sum_{0\le i\le p-1} Q_i(x_1,\ldots,x_{d-1})x_d^i$. We also introduce $\mathrm{red}^0(Q) = Q$, and for each $k \ge 0$, $\mathrm{red}^{k+1}(Q) = \mathrm{red}(\mathrm{red}^k(Q))$. Finally we let $\mathrm{der}(Q)$ denote the x_d-derivative of Q.

Let $A(x) = \sum_{0 \le i \le a} \alpha_i x^i$ and $B(x) = \sum_{0 \le i \le b} \beta_i x^i$ be two polynomials in the real variable x with $\deg(A) = a$ and $\deg(B) = b$. The *Sylvester matrix* of A and B is the $(a+b) \times (a+b)$ matrix M obtained by placing the coefficients of the polynomials $x^{b-1}A(x), \ldots, xA(x), A(x), x^{a-1}B(x), \ldots, xB(x), B(x)$ in consecutive rows of M, with the coefficients of x^i appearing in column $a+b-i$:

$$\begin{pmatrix}
\alpha_a & \alpha_{a-1} & \cdots & \alpha_0 & & & \\
 & \alpha_a & \alpha_{a-1} & \cdots & \alpha_0 & & \\
 & & \ddots & \ddots & \ddots & \ddots & \\
 & & & \alpha_a & \alpha_{a-1} & \cdots & \alpha_0 \\
\beta_b & \beta_{b-1} & \cdots & \beta_0 & & & \\
 & \beta_b & \beta_{b-1} & \cdots & \beta_0 & & \\
 & & \ddots & \ddots & \ddots & \ddots & \\
 & & & \beta_b & \beta_{b-1} & \cdots & \beta_0
\end{pmatrix}$$

The determinant of M is the *resultant* of A and B. For $0 \le j \le \min(a,b)$, M_j is the matrix obtained by deleting the last j rows of A coefficients, the last j rows of B coefficients, and all the last $2j$ columns. We can then define $\mathrm{psc}^j(A,B)$ (the j^{th} *principal subresultant coefficient* of A and B) as the determinant of M_j (see Collins, 1975, for details). As is shown in (Brown and Traub, 1971), the unique factorization theorem for polynomials implies that A and B have exactly j common roots (i.e., j is the degree of their greatest common divisor) if and only if j is the least index k for which $\mathrm{psc}^k(A,B) \neq 0$. This result is at the basis of the recursive construction of a *cad*.

Let $\mathcal{F} = \{P_1, \ldots, P_n\}$ be a set of n real d-variate polynomials ($d \ge 2$) with rational coefficients and norm-length $\le \ell$. As expected, the construction of an $\mathcal{F}$-invariant K proceeds recursively. It suffices to specify what arguments should be passed to the decomposition algorithm at the first recursive call. To do so, we define the *projection* of $\mathcal{F}$, denoted $\mathcal{G}$, as the union of $\mathcal{G}_2$, $\mathcal{G}_3$, and $\mathcal{G}_4$, with

(i) $\mathcal{G}_1 = \{ \mathrm{red}^k(P) \mid P \in \mathcal{F} \text{ and } k \ge 0 \text{ and } \deg(\mathrm{red}^k(P)) \ge 1 \}$,

(ii) $\mathcal{G}_2 = \{ \mathrm{ldcf}(P) \mid P \in \mathcal{G}_1 \}$,

(iii) $\mathcal{G}_3 = \{ \mathrm{psc}^k(P, \mathrm{der}(P)) \mid P \in \mathcal{G}_1 \text{ and } 0 \le k < \deg(\mathrm{der}(P)) \}$,

(iii) $\mathcal{G}_4 = \{ \mathrm{psc}^k(P,Q) \mid P, Q \in \mathcal{G}_1 \text{ and } 0 \le k < \min(\deg(P), \deg(Q)) \}$.

The notion of a *projection* generalizes the idea of (Dobkin and Lipton, 1976) of pairing up hyperplanes: the pairing takes place in $\mathcal{G}_4$; the set $\mathcal{G}_3$ accounts for multiple

roots (for polynomials in x_d), while $\mathcal{G}_2$ is included because a change in the number of roots can be caused by a loss of degree. The following result is proven in (Collins, 1975): let K' be a $\mathcal{G}$-invariant *cad* of E^{d-1} and let c_i be any cell of K'; the total number of distinct real roots of the polynomials in x_d, $P_1(x_1,\ldots,x_d),\ldots,P_n(x_1,\ldots,x_d)$, remains constant as $(x_1,\ldots,x_{d-1})$ varies in c_i. These roots form a well-ordered set of continuous functions over c_i: $f_{i,1}(x_1,\ldots,x_{d-1}),\ldots,f_{i,\nu_i}(x_1,\ldots,x_{d-1})$; in particular, no two such roots ever coincide over c_i. As a result, for each $c_i \in K'$, the partition of $c_i \times E^1$ induced by the hypersurfaces $x_d = f_{i,1}(x),\ldots,x_d = f_{i,\nu_i}(x)$ $(x \in E^{d-1})$ defines an $\mathcal{F}$-invariant *cad* of E^d.

This provides a recursive scheme for computing an $\mathcal{F}$-invariant *cad* K of E^d. The algorithm takes $\mathcal{F}$ and d as input and recurses by calling itself with $\mathcal{G}$, the projection of $\mathcal{F}$, and $d-1$ as arguments. The output of Collins' construction includes (i) quantifier-free formulas defining each cell of K and (ii) a *cas* of K of the form $\{\beta_1,\ldots,\beta_\nu\}$, where for each $i = 1,\ldots,\nu$, each coordinate of $\beta_i \in E^d$ is represented by a quantifier-free formula. For our application the definitions of the cells are not really needed: instead, we need a correspondence between sample points and their associated polynomials in $\mathcal{F}$. If $d > 1$ then K has a base *cad* $K' = \{c_1,\ldots,c_\mu\}$ which is $\mathcal{G}$-invariant. Let $\{\beta'_1,\ldots,\beta'_\mu\}$ be the *cas* of K', computed recursively. For each $i = 1,\ldots,\mu$, let $\{\beta_{i,1},\ldots,\beta_{i,2\nu_i+1}\}$ be the points of the *cas* of K, ordered in ascending x_d-order, whose first $d-1$ coordinates form the point β'_i. Each point $\beta_{i,2j}$ $(1 \le j \le \nu_i)$ lies on at least one algebraic variety of the form $P_l(x) = 0$. Let $l_{i,j}$ be any such value of l and let $\beta_{i,2j} = (a_1,\ldots,a_d)$; we define $m_{i,j}$ as the number of distinct real roots of $Q(y)$ that are strictly smaller than a_d, where $Q(y) = P_{l_{i,j}}(a_1,\ldots,a_{d-1},y)$ is regarded as a polynomial in y. As part of the output, we require the sequence $\{(l_{i,1},m_{i,1}),\ldots,(l_{i,\nu_i},m_{i,\nu_i})\}$ for each $i = 1,\ldots,\mu$. This sequence will be necessary later on in order to carry out the binary searches underlying the point location algorithm.

The next step is to show how to derive these sequences from $\{\beta_{i,1},\ldots,\beta_{i,2\nu_i+1}\}$ $(1 \le i \le \mu)$. Recall that the latter sequences are provided directly by the Collins construction. Let $\phi(x)$ be the quantifier-free defining formula for $\beta_{i,2j}$ $(1 \le j \le \nu_i)$. Trivially, we can test the predicate

$$[\exists x \in E^d \mid \phi(x) \text{ and } P_l(x) = 0]$$

for each $l = 1, \ldots, n$, and pick as $l_{i,j}$, say, the first value of l found to satisfy the predicate. To obtain $m_{i,j}$ it suffices to express with a prenex formula the proposition, denoted F_k, that z is a root of Q and $Q(y) = P_{l_{i,j}}(a_1, \ldots, a_{d-1}, y)$ has exactly k distinct roots strictly smaller than z. In the spirit of (Arnon, 1981) we express F_k with the formula

$$R^{l_{i,j}}_{a_1,\ldots,a_{d-1},k}(z) = (\exists y_1, \ldots, y_k)\,(\forall x)\Big| \Big[(Q^2(z) + Q^2(y_1) + \cdots + Q^2(y_k) = 0) \text{ and } (y_1 < \cdots < y_k < z) \text{ and } (Q(x) \neq 0 \text{ or } z \leq x \text{ or } \prod_{1 \leq i \leq k} (y_i - x) = 0)\Big].$$

The value of $m_{i,j}$ is then given by the unique index k for which $R^{l_{i,j}}_{a_1,\ldots,a_{d-1},k}(z)$ is true, with $\beta_{i,2j} = (\beta'_i, z)$.

IV) *Complexity Analysis*: We assume that only rational symbolic calculations are used during the course of the computation. The following complexity results are derived from (Collins, 1975). Let b be the maximum degree of any polynomial in $\mathcal{F}$ in any variable. Recall that all the polynomials in $\mathcal{F}$ have norm-length at most ℓ. We assume that d and b (but *not necessarily* ℓ) are constants. The $\mathcal{F}$-invariant *cad* produced by the Collins construction consists of $O\left((2b)^{3^{d+1}} n^{2^d-1}\right) = O\left(n^{2^d-1}\right)$ cells. (Collins' paper actually states a slightly larger bound, but the one above easily follows from his derivations.) The total number of polynomials defined in the various projections introduced in the decomposition is bounded above by $O\left((2b)^{3^d} n^{2^{d-1}}\right) = O\left(n^{2^{d-1}}\right)$ and the maximum degree of each polynomial in any variable is at most $\frac{1}{2}(2b)^{2^{d-1}} = O(1)$. The norm-length of each polynomial is at most $(2b)^{2^d}\ell = O(\ell)$.

Consider now the *cas* of the decomposition. Each algebraic point is represented by its coordinates. Collins uses two different representations of real algebraic numbers. One is the traditional root isolation method: the number α is the unique real root of an integral polynomial falling in some interval I, whose endpoints are rationals of the form $a/2^k$. In the other representation, a real algebraic number β will appear as an element of the algebraic number field $Q(\alpha)$ (i.e., the smallest subfield of $\Re$ that contains both Q and α). In this case, we represent β by a rational polynomial $B(x)$ with $\beta = B(\alpha)$. The degree of each polynomial used in the definition of the *cas*'s is dominated by $(2b)^{2^{2d-1}} = O(1)$ and their norm-length is at most $\ell(2b)^{2^{2d+3}} n^{2^{d+1}} = O\left(\ell n^{2^{d+1}}\right)$. Implementing the

Collins construction proper requires $O\left(\ell^3(2b)^{2^{2d+8}}n^{2^{d+6}}\right) = O\left(\ell^3 n^{2^{d+6}}\right)$ bit operations. Using Theorem 1 and the previous upper bounds, it is easy to see that this running time asymptotically dominates the overhead of computing the sequences of the form $\{(l_{i,1}, m_{i,1}), \ldots, (l_{i,\nu_i}, m_{i,\nu_i})\}$. In the unit-cost model, this gives us a total running time of $O\left(n^{2^{d+6}}\right)$.

3. The Generalized Point Location Algorithm

Most of the ingredients entering the composition of the algorithm have already been introduced. The data structure $\mathcal{D}(\mathcal{F})$ is defined recursively as follows: it includes

(i) $\mathcal{D}(\mathcal{G})$, where $\mathcal{G}$ is the projection of $\mathcal{F}$;

(ii) a *cas* of K;

(iii) a set of ν one-word memory cells $C_1, \ldots, C_\nu$ (which we conveniently associate with the cells of K).

Let $C_1^g, \ldots, C_\mu^g$ be the memory cells associated with $\mathcal{D}(\mathcal{G})$ (in one-to-one correspondence with the cells of $K' = \{c_1, \ldots, c_\mu\}$). Each cell C_i^g $(1 \leq i \leq \mu)$ stores a pointer to the sequence $\{l_{i,1}, \ldots, l_{i,\nu_i}\}$ previously defined. Recall that the cell C_i^g is associated with $2\nu_i + 1$ cells of K (each projecting exactly onto c_i). Let $W_i = \{C_{i,1}, \ldots, C_{i,2\nu_i+1}\}$ be the corresponding memory cells in ascending x_d-order. Consider the sequence $S_i = \{l_{i,1}, \ldots, l_{i,\nu_i}\}$ as an ordered set of *keys*. The possible outcomes of a binary search in this set form a sequence of $2\nu_i + 1$ keys and open intervals, which we put in one-to-one correspondence with W_i. The data structure is now complete, so we can describe the algorithm.

The input is a family of polynomials $\mathcal{F}$, assumed to be preprocessed as previously described. The *generalized point location* problem defined earlier can be reduced to the following: given a query point $x = (x_1, \ldots, x_d) \in E^d$, compute the index i such that C_i corresponds to the unique cell of K that contains x. If x is a zero of one of several polynomials of $\mathcal{F}$, the index of one of them will be directly available from C_i. If, on the other hand, C_i lies in one of the connected regions of $\mathcal{C} = \left\{ x \in E^d \mid \prod_{1 \leq i \leq n} P_i(x) \neq 0 \right\}$, access to the sample points provided by the *cas* of K would provide the desired answer. But often the data structure does not need to store the sample points. Recall that we are interested in evaluating a particular function f which is invariant over the regions of

C. Before throwing away the sample points, we will precompute and store the values of f at these points. Note that from a theorem of (Milnor, 1964) the number of connected regions in C is at most singly exponential in d. Therefore the evaluation of f over the sample points is bound to produce the same values repeatedly.

If $d = 1$ the algorithm is a trivial binary search, so let us assume that $d > 1$. Recursively, we assume that we have available the index of the cell C_i^g that contains $(x_1, \ldots, x_{d-1})$. Perform a binary search in S_i with respect to x_d, and report the element of W_i corresponding to the result of the search. We can implement the generic comparison against $l_{i,j}$ as the two-fold question:

1. Does $P_{l_{i,j}}(x) = 0$ and is x the $(m_{i,j}+1)$st real root of $P_{l_{i,j}}$?
2. Is x_d strictly larger or smaller than the $(m_{i,j}+1)$st real root of $P_{l_{i,j}}(x_1, \ldots, x_{d-1}, y)$, regarded here as a polynomial in y?

The first part of Question (1) is easy to answer since it involves a simple polynomial evaluation. The second part requires evaluating the predicate R. The second question can be answered by computing the predicate

$$\left[(\forall y)\,|\,(y > x_d) \text{ or } \neg\left(R^{l_{i,j}}_{x_1,\ldots,x_{d-1},m_{i,j}+1}(y)\right)\right].$$

Of course, since the polynomial $A(y) = P_{l_{i,j}}(x_1, \ldots, x_{d-1}, y)$ has constant degree and rational coefficients it might be just as simple to enumerate all its real roots (as algebraic numbers) and compare x_d against them. Each polynomial occurring in any projection has degree $O(1)$ and norm-length $O(\ell)$, so the analysis above applies to all the binary searches performed in the location of x. Since the total number of these polynomials is in $O(n^{2^{d-1}})$, the overall query time amounts to $O(\ell^3 \log n)$ in the bit model and $O(\log n)$ in the unit-cost model. (Incidentally, note that these bounds involve constant factors doubly exponential in d.) The preprocessing time is $O\left(n^{2^{d+6}}\right)$ in the unit-cost model and $O\left(\ell^3 n^{2^{d+6}}\right)$ in the bit model.

Theorem 2. *Let $\mathcal{F} = \{P_1, \ldots, P_n\}$ be a family of n d-variate polynomials with rational coefficients and constant maximum degree. The generalized point location problem on $\mathcal{F}$ can be solved in $O(\log n)$ query time, using a data structure of size $O\left(n^{2^d-1}\right)$; the preprocessing time is $O\left(n^{2^{d+6}}\right)$. This assumes that operations on any integers of length proportional to the norm-length of the polynomials of $\mathcal{F}$ can be done in constant time.*

4. Point Location and Multidimensional Searching

Multidimensional searching refers to the general task of querying a database to retrieve information of a particular nature. This can be defined formally by introducing a finite set V, a query space Q, and a response domain R. We also need a predicate function $p : Q \times V \rightarrow \{0,1\}$ and an *evaluation* function $c : 2^V \rightarrow R$. A query is an arbitrary element $q \in Q$ and its output is the value of $c(\{ x \in V \mid p(q,x) \})$. A classical example is *orthogonal range searching*: V is a set of points in E^d, Q is the set of all d-dimensional isothetic hyperrectangles, $p(q,x)$ is true if and only if the point x lies in the hyperrectangle q, and c returns the cardinality of the input set in the counting version of the problem; in the reporting version, c is the identity function.

Elements of both V and Q are expressed as points in Euclidean space, and the value of the predicate $p(q,x)$ is determined by the signs of certain polynomials $P_x, Q_x, \ldots$ evaluated at q. The family $\mathcal{F} = \{ P_x, Q_x, \ldots \mid x \in V \}$ is assumed to consist of d-variate polynomials of constant maximum degree with rational coefficients. The reduction of multidimensional searching to point location is now obvious. This is called the *locus approach*: subdivide E^d via the varieties defined by $\mathcal{F}$ and assign to each resulting cell the corresponding (constant) value of the function c.

Two implementations of the locus approach suggest themselves. One is to apply the point location algorithm of the previous section. The other approach, suggested in (Yao and Yao, 1985), is to linearize the polynomials by throwing in additional variables. For example, the variety in E^3,

$$xy^2z^3 + 2x^2y^3 - 3z^4 + x^2y + z = 0$$

can be replaced by the hyperplane in E^5

$$z_1 + 2z_2 - 3z_3 + z_4 + z_5 = 0.$$

Searching for the location of the point (x,y,z) is thus reduced to the point location of $(xy^2z^3, x^2y^3, z^4, x^2y, z) \in E^5$ with respect to a linear variety. The obvious disadvantage of the latter method is that the number of variables may jump from, say, d, to $(b+1)^d$, where b is the maximum degree in a single variable of any polynomial of $\mathcal{F}$. This can have dire consequences, as the preprocessing cost will be doubly exponential in d in one case and doubly exponential in $(b+1)^d$ in the other.

Let us illustrate our point location approach on a specific example. Let V be a set of n points in E^d and Q be the set of d-variate polynomials of degree at most b with rational coefficients. Given a query polynomial $q \in Q$, count the number of points $x \in V$ such that $q(x) > 0$. In our framework the query q can be regarded as a rational point in E^c, where c is the dimension of the vector space Q. It is well-known that $c = \binom{d+b}{b}$. (The dimension c is equal to the number of ways one can assign exponents to $x_1, x_2, \ldots, x_d$ adding up to at most b.) The family $\mathcal{F}$ consists of n linear forms of c variables. A query is answered in time $O(\log n)$ at the cost of $O(n^{2^c-1})$ space.

Consider now the case where V is a set of n points in E^d and a query is a pair (q, r) consisting of a point $q \in E^d$ and a positive rational number r. The response to the query is the number of points in V lying within a distance r of q. The family $\mathcal{F}$ consists of n $(d+1)$-variate polynomials $p(x_1, \ldots, x_{d+1})$ of the form $(x_1-a_1)^2+\cdots+(x_d-a_d)^2-x_{d+1}^2$. A query is answered in time $O(\log n)$ at the cost of $O(n^{2^{d+1}-1})$ space.

5. Biggest Stick, Segment Shifting, and Other Related Problems

In this section we concern ourselves with the following class of problems: given two collections A, B of n objects each and a real-valued function F defined on $A \times B$, compute the minimum of F over $A \times B$. If the function F can be evaluated anywhere in constant time, problems of this type always have trivial $O(n^2)$ solutions. Note that many common problems fall in this category, e.g., *Hopcroft's problem* (given a collection of lines and points in the plane, determine whether any line passes through any point) and the *diameter problem* in E^3 (given a three-dimensional polytope, compute the largest interdistance between any two vertices).

We will give a method for solving these problems in subquadratic time. The technique is very general, and will always work as long as a fixed number of rational parameters are needed to represent objects in A or B, and the expression $F(a, b)$ can be specified by a straight-line program of constant length involving algebraic functions (in the parameters specifying a and b) of bounded degree. Rather than describing the method in full generality we will illustrate it by looking at two problems posed in the literature. In a different context the next section will also provide an example of the same basic technique.

Here is a problem posed in (McKenna, 1986): given two collections F and G of nonvertical segments in E^3, such that each segment in F (resp. G) is parallel to the xz-plane (resp. the yz-plane) and each segment in F lies above every segment in G, find the largest distance d by which F can be shifted downwards until it hits G. It is easy to rephrase this problem in the framework outlined above. As it turns out, it is not much more difficult to solve the more general *segment shifting problem* obtained by removing any restriction on the orientation of the segments.

As a starter, we consider a special case of the problem, *restricted line shifting*, where both F and G are collections of infinite lines. Each line of F is of the form $(y = y_i,\ z = a_i x + b_i)$, for $i = 1, \ldots, m$, and each line of G is of the form $(x = x_j,\ z = c_j y + d_j)$, for $j = 1, \ldots, n$. In this particular case we want to compute

$$\min_{i,j}(a_i x_j + b_i - c_j y_i - d_j).$$

Put $u_i = (a_i, b_i, y_i, 1)$, for $i = 1, \ldots, m$ and $v_j = (x_j, 1, -c_j, -d_j)$, for $j = 1, \ldots, n$. We must now compute $\min_{i,j} u_i \cdot v_j$, which gives us a new problem.

Given two sets of vectors $U = \{u_1, \ldots, u_m\}$ and $V = \{v_1, \ldots, v_n\}$ in E^d, find

$$\min(u_i \cdot v_j \mid 1 \le i \le m,\ 1 \le j \le n).$$

Clearly the minimum must be attained at points u_i and v_j lying on the convex hulls of U and V, respectively. So, without loss of generality, suppose that all points u_i, v_j lie on the corresponding hulls. (Note that for the set of vectors arising in the restricted line shifting problem, computation of the convex hulls can be done in time $O(m \log m)$ and $O(n \log n)$, respectively, because each of these sets lies in a 3-dimensional cross-section of E^4.) Next, without loss of generality, assume that $m \le n$. For each vector v, the minimum of $u_i \cdot v$ is attained at that point (or points) $u_i \in U$ at which U is supported by a hyperplane whose inward drawn normal is v. Therefore, we need to preprocess U into a data structure that supports queries of the above form. This is reasonably easy to do in E^2 and E^3. In E^3, for example, we use the standard Gaussian sphere representation, also known as the normal diagram of U. That is, we define a map on the sphere S^2 with $O(n)$ regions, so that for each region R_i there corresponds a vector $u_i \in U$ such that all planes with inward drawn normals in R_i support U at u_i. Next, we construct a data

structure which supports $O(\log m)$ point location queries in this spherical map. Such a data structure can be obtained in $O(m \log m)$ preprocessing. We can now determine $\min_i(u_i \cdot v)$ in logarithmic time by simply locating v in the map. In a total time of $O((m+n)\log\min(m,n))$ we can thus find the required minimum.

Note that this approach is also applicable to the restricted line shifting problem, because the underlying set is essentially 3-dimensional. We can thus preprocess U as above; given any vector v we simply remove its fourth coordinate and find the plane supporting U whose inward normal is precisely the truncated v. This gives us an $O((m+n)\log\min(m,n))$ time solution to McKenna's restricted line shifting problem.

Next, we turn to the general segment shifting problem. Let A and B be two collections, each consisting of n nonvertical segments in E^3. The problem is to find the smallest positive vertical distance $F(a,b)$ between any pair of segments $a \in A$ and $b \in B$. The function $F(a,b)$ is defined as follows: if the projections of a and b onto the xy-plane intersect at some point c, then $F(a,b)$ is equal to $z_a - z_b$, where z_a (resp. z_b) is the z-coordinate of the point of a (resp. b) projecting into c; otherwise $F(a,b) = +\infty$.

Fix two segments $a \in A$, $b \in B$. Each of them can be specified by six parameters, e.g., the coordinates of its two endpoints. Let a_1, a_2 be the endpoints of a and b_1, b_2 be the endpoints of b. Let $a_1^*, a_2^*, b_1^*, b_2^*$ be the projections of these four points onto the xy-plane. We first find the two rational parameters α, β, satisfying

$$a_1^* + \alpha(a_2^* - a_1^*) = b_1^* + \beta(b_2^* - b_1^*).$$

For the projections of a and b to intersect, it is necessary and sufficient that $0 \le \alpha \le 1$ and $0 \le \beta \le 1$. Clearly both α and β are rational functions of a_1^*, a_2^*, b_1^*, and b_2^*. Once α and β have been found (and lie between 0 and 1), the desired $F(a,b)$ is equal to

$$Z(a,b) = a_1^z + \alpha(a_2^z - a_1^z) - b_1^z - \beta(b_2^z - b_1^z),$$

where a_1^z, a_2^z, b_1^z, and b_2^z are the z-coordinates of the four corresponding points.

In other words, regarding each segment $b \in B$ as a point in E^6, for each $a \in A$, we can express $F(a,b)$ as follows:

$$F(a,b) = \begin{cases} Z(a,b), & \text{if } 0 \le \alpha(a,b) \le 1 \text{ and } 0 \le \beta(a,b) \le 1; \\ +\infty, & \text{otherwise,} \end{cases}$$

where $\alpha(a,b)$, $\beta(a,b)$, and $Z(a,b)$ are all rational functions of a and b. Now consider the collection $\mathcal{F}$ of $O(n^2)$ rational functions consisting of $\alpha(a,b)$, $1-\alpha(a,b)$, $\beta(a,b)$, $1-\beta(a,b)$, and $Z(a,b) - Z(a',b)$, for $a, a' \in A$. If we now apply the point location algorithm to the collection $\mathcal{F}$ we will be able to compute $\min_{a \in A} F(a,b)$ by simply locating b in its proper Collins cell.

What does that presuppose? First of all, Theorem 2 tells us that the collection $\mathcal{F}$ should consist of polynomials and not rational functions. This is clearly not a problem: any rational P/Q can be replaced in $\mathcal{F}$ by the two polynomals P and Q, thus at most doubling the size of the collection. Secondly, we need to have an explicit correspondence between each cell of the Collins decomposition and its associated "winning" segment of A (if any). To obtain one, the naive solution will do: simply interpret each point of the *cas* as a segment in E^3 and test it against every segment of A. Break ties arbitrarily. What is the complexity of this solution? The *cas* consists of $O\left((n^2)^{2^6-1}\right) = O(n^{2^7-2})$ algebraic points, so labeling the Collins cells with the proper segments of A will require $O(n^{2^7-1})$ time. This is largely dominated by the estimate of the preprocessing time given by Theorem 2, that is, $O\left((n^2)^{2^{12}}\right) = O(n^{2^{13}})$.

We can now use these observations to obtain a subquadratic solution to the segment shifting problem. We use a batching strategy inspired by Yao's work on higher-dimensional minimum spanning trees (Yao, 1982). The technique is somewhat reminiscent of the Four-Russian algorithm for Boolean matrix multiplication (Aho et al., 1974); see also (Chazelle, 1987) for a generalized version of it. The idea is to partition the collection A into $\lceil n^{1-1/2^{13}} \rceil$ subsets of roughly equal size. Instead of preprocessing the entire set A for point location, as decribed above, we work on each subset A_i separately: for every segment $x \in B$ we will determine in logarithmic time the minimum distance to A_i. Repeating this operation for each subset A_i gives an overall running time of $O(n^{1-1/2^{13}} n \log n) = O(n^{2-1/2^{13}} \log n)$.

Theorem 3. *The segment shifting problem on n line segments can be solved in* $O(n^{1.999878})$ *time.*

We shall follow a similar approach to solve another problem posed by (McKenna, 1986): given a simple n-gon P, what is the longest line segment that can be drawn in the closure of P? In McKenna's terminology the segment is called the ***biggest stick*** of P.

Obviously, the biggest stick is not necessarily unique (think of a regular n-gon), so the term actually refers to *any* segment with the characteristics described above. A simple yet crucial observation of McKenna is that any biggest stick must pass through two distinct vertices of P. Going through n iterations of a linear vertex-visibility algorithm (ElGindy and Avis, 1981), an $O(n^2)$ solution to the biggest stick problem follows readily. If the vertex-visibility graph is $o(n^2)$ then we can obtain better results by using output-sensitive methods for computing vertex visibilities, as in (Hershberger, 1987, Ghosh and Mount, 1987, Kapoor and Maheshwari, 1988). Of course, in the worst case the complexity is still quadratic. Can one do better?

To begin with, we set the stage for divide-and-conquer by applying the polygon-cutting theorem (Chazelle, 1982). In $O(n \log n)$ time we find a diagonal c which partitions P into two subpolygons, P_1 and P_2, each of size at least roughly $n/3$. The diagonal c is a line segment inside P joining two of its vertices. Next, we call the algorithm recursively to determine a biggest stick in each of P_1 and P_2. What remains to be done is to find a biggest stick crossing the diagonal c and keep the biggest of all three as the output. To look at the problem in dual space will clarify some of the issues.

Consider the dual mapping which puts in one-to-one correspondence the point (a, b) and the line $ax+by+1 = 0$. If d is the distance from the origin O to the point p, the dual of p is the line perpendicular to Op at distance $1/d$ from O and placed on the other side of O. Any line ℓ crossing c is thus mapped to a point ℓ^* in the dual plane. This point lies in the double wedge W (not containing the origin) formed by the dual lines of the endpoints of c. For each such line and $i = 1, 2$, let $F_i(\ell^*)$ be the length of the connected portion of $\ell \cap P_i$, one of whose endpoints lies on c. Clearly the length of a biggest stick s^* through c is given by

$$\max\{\, F_1(\ell^*) + F_2(\ell^*) \mid \ell \text{ crosses } c \,\}.$$

As shown in (Chazelle and Guibas, 1989), each of the functions F_1 and F_2 can be represented as a piecewise smooth function such that the projection of its smooth portions forms a straight-edge convex subdivision of W. The domain of the functions can be extended to the whole plane (setting F_i to 0 outside of W), which gives us two convex subdivisions M and N of the plane. These subdivisions encode the set of boundary points of P that are visible from the diagonal c. More precisely, an edge of M (resp. N)

encodes the visibility of a vertex of P_1 (resp. P_2) from c; a vertex of M (resp. N) is dual to a line passing through c and two vertices of P_1 (resp. P_2) with no other contact with P between these three intersections. It is shown in (Chazelle and Guibas, 1989) that both M and N have $O(n)$ vertices and can be computed in $O(n \log n)$ time. McKenna has observed that $F_1(\ell^*) + F_2(\ell^*)$ is maximized either at a vertex of M or N or at an intersection between an edge of M and an edge of N. Finding the biggest stick s^* can now be regarded as a special case of the following more general problem.

Given two bivariate functions $F(x, y)$ and $G(x, y)$ and two associated convex planar subdivisions M and N, such that F (resp. G) is smooth (or continuous, or convex) over each region of M (resp. N), and such that $F + G$ attains its maximum either at vertices of M or N or at intersections of edges of M with edges of N, is it possible to determine the maximum of $F + G$ in $o(mn)$ time, where m (resp. n) denotes the number of vertices of M (resp. N)? By preprocessing each subdivision for point location we can evaluate F and G at any point in $O(\log n + \log m)$ time. This allows us to evaluate $F + G$ at all the vertices of M and N in $O(n \log n + m \log m)$ time. There now remains the more difficult task of testing edges against each other.

It should be clear that our previous batching technique can be applied in much the same way. A few differences are worth noticing, however. Let us discuss the problem of computing $\max_{(x,y)\in e} F(x, y) + G(x, y)$ in logarithmic time, given some edge e of, say, N. First, we must recall the geometric meaning of the subdivision N. With each edge e of N is associated a vertex v of P_2 as well as a line segment ℓ on the boundary of P_2 (Figure 1).

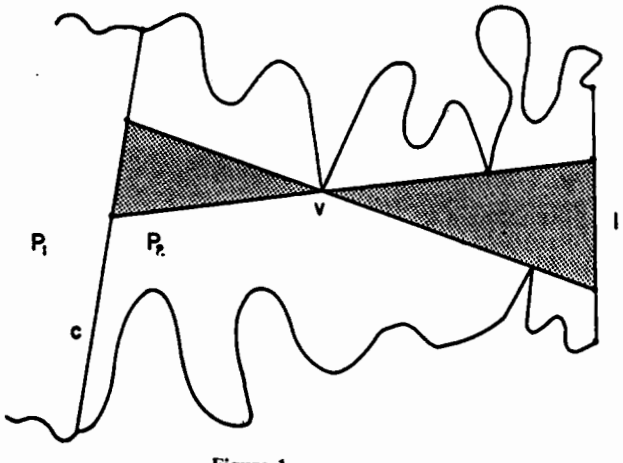

Figure 1

For this reason, e can be represented as a point $z \in E^6$ (two coordinates for v and four for the segment ℓ). Note that each point of $e = e(z)$ is mapped dually to a line passing through both the diagonal c and the segment ℓ. The edge $e(z)$ is "obtained" by pivoting the line in question around v and scanning all of ℓ. Given two edges $a \in M$ and $e(z) \in N$, we define $H(a,z)$ as being 0 if a and $e(z)$ do not intersect and the value of $F(x,y) + G(x,y)$, where $(x,y) = a \cap e(z)$, otherwise. Given an edge $e(z)$ of N we compute $\max_{a \in M} H(a,z)$ by performing a point location in the appropriate *cad* in E^6. Which functions should be included in the underlying collection $\mathcal{F}$? Let $\big(p(a,z), q(a,z)\big)$ be the intersection of a and $e(z)$. Since the endpoints of the edge $e(z)$ can be expressed as rational functions of z (i.e., of its coordinates), so can $p(a,z)$ and $q(a,z)$. Note that the intersection of a and $e(z)$ can be ensured by enforcing the signs of four three-by-three determinants, each also a rational function of z. These constraints ensure that the endpoints of a and $e(z)$ lie on opposite sides of the intersection of the lines containing the segments.

Let L be the line dual to the point $\big(p(a,z), q(a,z)\big)$ and let A (resp. B) be the intersections of L with the portion of the boundary of P_1 (resp. P_2) associated with the edge a (resp. $e(z)$). If $H(a,z)$ is not zero then it is equal to the length of the segment $|AB|$. The idea is to include all functions $H(a,z) - H(b,z)$ in $\mathcal{F}$, for all edges $a, b \in M$. A minor problem is that $H(a,z)$ is not a polynomial, but the square root of a rational function, and it is not even continuous. It is easy to extend H continuously, however, by defining it as $F(x,y)+G(x,y)$, where (x,y) is now the intersection of the *lines* supporting a and $e(z)$. Then, we include in $\mathcal{F}$ the rational functions $\big(H(a,z)\big)^2 - \big(H(b,z)\big)^2$. We also add to $\mathcal{F}$ the determinants that test for intersections between $e(z)$ and edges of M.

Let us mention in passing a general method for dealing with sets $\mathcal{F}$ that include arbitrary algebraic functions (and not just polynomials). Suppose that we have a function $f(x_1, \ldots, x_k)$ which is expressed as a root of a univariate polynomial $P(z)$ whose coefficients are polynomials in $x_1, \ldots, x_k$. Any zero $(x_1, \ldots, x_k)$ of f is also a zero of the polynomial $P(0)$, so we can replace f by $P(0)$ in $\mathcal{F}$. As long as the queries are not zeros of $P(0)$ themselves the Collins decomposition obtained after the replacement will work just fine. For example, let $f(x,y,z) = \sqrt{x} - 2\sqrt{y} + 3\sqrt{z}$. Using repeated squaring, we derive the identity

$$\big((f^2 + x - 4y - 9z)^2 - 4xf^2 - 144yz\big)^2 = 2304xyzf^2.$$

Therefore we can use the substitute function

$$x^2 + 16y^2 + 81z^2 - 8xy - 72yz - 18xz.$$

Returning now to the biggest stick problem, we are about ready to conclude. Once again the cost of precomputing the function on the algebraic points of the *cas* is dominated by the construction of the data structure. As in the segment shifting problem, $\mathcal{F}$ contains $O(n^2)$ functions, therefore the preprocessing requires $O(n^{2^{13}})$ time. The batching trick leads to the following result.

Theorem 4. *Computing the biggest stick of a simple n-gon can be done in* $O(n^{1.999878})$ *time.*

Note that the same basic technique can be applied to handle the sum of two bivariate functions over more general planar maps. The complexity, of course, will depend on the particular definitions of these functions.

6. A Problem on Points in Motion

Atallah has posed the following problem (Atallah, 1985): Suppose that n points are moving in the plane, with the trajectory of each point described by a polynomial function of the time. What is the first instant at which their convex hull will enter a steady-state, i.e., a combinatorially stable configuration? We shall assume that each polynomial has degree bounded above by a constant. Since the output can be a fairly arbitrary algebraic number, we will content ourselves with a description of that number involving a defining polynomial along with an isolating interval. The naive algorithm consists of computing the steady convex hull in $O(n \log n)$ time (Atallah, 1985), and then determining the first time each point will achieve its steady positioning with respect to each edge on the hull. Let e be an edge of the hull: for each moving point p it suffices to compute the last instant (if any) at which p lies on the line supporting the edge e. Let $t(e)$ be the maximum value obtained by this process. The desired answer is the maximum value of $t(e)$ over all edges e of the convex hull. Can this quadratic algorithm be improved? We will show that it can indeed—well, at least theoretically. We will use our generalized point location algorithm to produce an $O(n^{2-\epsilon})$ time algorithm, for some small positive constant ϵ.

Let $V = \{p_1, \ldots, p_n\}$ be a set of $n > 2$ moving points in the Euclidean plane. We assume the existence of $2n$ univariate polynomials $x_1, y_1, \ldots, x_n, y_n$ of degree d with rational coefficients, such that for each i $(1 \le i \le n)$, $x_i(t)$ and $y_i(t)$ are respectively the x and y coordinates of p_i at time $t \ge 0$. Let $p_{i_1}, \ldots, p_{i_k}$ be the points on the boundary of the convex hull of V at time t, given in clockwise order, with $i_1 < \min(i_2, \ldots, i_k)$. (If somehow $p_{i_j}, \ldots, p_{i_{j'}}$ coincide for some j, j' then their indices should appear in the order $i_j < \ldots < i_{j'}$.) Let $H(t)$ be the uniquely defined sequence $(i_1, \ldots, i_k)$. It is clear that $H(t)$ converges, as t grows to infinity [A]. We define the *threshold* of $H(+\infty)$ as the smallest value of $t \ge 0$ such that $[(\forall t' \ge t) \mid H(t) = H(t')]$.

Let $x_i(t) = \sum_{0 \le j \le d} a_{i,j} t^j$ and $y_i(t) = \sum_{0 \le j \le d} b_{i,j} t^j$, for $i = 1, \ldots, n$. Without loss of generality, assume that $H(+\infty)$ is the sequence $(1, \ldots, m)$, for $m \le n$, and that all n points $(a_{i,0}, \ldots, a_{i,d}, b_{i,0}, \ldots, b_{i,d})$ of Q^{2d+2} are pairwise distinct. In $O(n \log n)$ time compute $H(+\infty)$ [A] and check all pairs (p_i, p_{i+1}) $(1 \le i \le m)$ in order to determine the largest $t_0 \ge 0$ such that, for some i, we have $x_i(t_0) = x_{i+1}(t_0)$ and $y_i(t_0) = y_{i+1}(t_0)$; if not defined, set $t_0 = -\infty$. Here (as in the following), index arithmetic is taken mod m. Similarly, we ensure the convexity of the polygon $\{p_1, \ldots, p_m\}$ by considering the function

$$f_i(q) = \big(y_i(t) - y_{i+1}(t)\big)q_x + \big(x_{i+1}(t) - x_i(t)\big)q_y + x_i(t)y_{i+1}(t) - y_i(t)x_{i+1}(t).$$

The point $q = (q_x, q_y) \in E^2$ lies to the right (resp. on or to the left) of the oriented line $(p_i, \overrightarrow{p_i p_{i+1}})$ iff $f_i(q) < 0$ (resp. $f_i(q) = 0$ or $f_i(q) > 0$). For each p_i $(1 \le i \le m)$ compute the largest real root of $f_{i+1}(p_i)$ as a polynomial in t; discard every case where the polynomial is identically zero. Let t_1 be the largest value thus obtained (or $-\infty$ if there is none), and let $t_2 = \max(0, t_0, t_1)$. Once $H(+\infty)$ is available, t_2 can be easily computed in $O(n)$ time. All that remains to be done is to compute the first instant at which each p_j $(m < j \le n)$ lies inside $H(+\infty)$ for good. To accomplish this, we allow ourselves some preprocessing. Let $q(t) = (q_x(t), q_y(t))$ be a time-varying point in E^2, with $q_x(t) = \sum_{0 \le j \le d} q_j^x t^j$ and $q_y(t) = \sum_{0 \le j \le d} q_j^y t^j$. The point $\chi = (q_0^x, \ldots, q_d^x, q_0^y, \ldots, q_d^y)$ belongs to E^{2d+2} and is independent of n. Let $\operatorname{sign} A = -1$ (resp. $= 0, 1$) if $A < 0$ (resp. $A = 0, > 0$). We define

$$t(\chi) = \min\Big\{ t \in \Re \,\Big|\, t \ge t_2 \text{ and } (\forall i;\, 1 \le i \le m)(\forall t' > t) \operatorname{sign} f_i\big(q(t)\big) = \operatorname{sign} f_i\big(q(t')\big) \Big\}.$$

Clearly, $t(\chi)$ can be computed in $O(m)$ time. Next, we describe a fast algorithm for computing $t(\chi)$ based on point location.

Let $\mathcal{F} = \{\phi_1(\chi,t),\ldots,\phi_m(\chi,t)\}$, where $\phi_i(\chi,t)$ denotes the $(2d+3)$-variate polynomial of degree $2d+1$,

$$f_i(\sum_{0\le j\le d} q_j^x t^j, \sum_{0\le j\le d} q_j^y t^j),$$

with t being the $(2d+3)$rd coordinate. Let K be the $\mathcal{F}$-invariant *cad* of E^{2d+3} provided by the procedure described in section 2, and let $K' = \{c_1,\ldots,c_\mu\}$ be its base *cad* (i.e., the induced *cad* of E^{2d+2}). Recall that, for each $c_i \in K'$, the procedure provides us with a sequence of indices (possibly empty) $S_i = \{l_{i,1},\ldots,l_{i,\nu_i}\}$ with the following meaning: for any given $\chi \in c_i$ the line $\chi \times E^1$ contains an increasing sequence of real roots for the univariate polynomials $\phi_{l_{i,1}}(\chi,t),\ldots,\phi_{l_{i,\nu_i}}(\chi,t)$. The interpretation of this sequence is trivial: it gives the indices i of the lines supporting p_ip_{i+1} that are intersected by the trajectory of χ in chronological order (from $t=-\infty$ to $t=+\infty$). If the sequence is empty then χ never intersects such a line. Once K' has been preprocessed for point location, computing $t(\chi)$ is straightforward. Locate the cell c_i that contains χ and check whether the sequence S_i is empty. If yes, set $t(\chi) = t_2$. If the sequence is not empty, the trajectory of χ intersects the line passing through $p_{l_{i,\nu_i}}p_{l_{i,\nu_i}+1}$ at some time t and does not intersect any other such line subsequently. We obtain t by computing the largest real root of $\phi_{l_{i,\nu_i}}(\chi,t)$ as a polynomial in t (which must exist). Finally we set $t(\chi) = \max(t_2,t)$. From Theorem 2 we immediately conclude that in $O(m^{2^{2d+9}})$ time it is possible to construct a data structure so that the function $t(\chi)$ can be evaluated at any point $\chi \in E^{2d+2}$ in $O(\log m)$ time.

We are now ready to attack Atallah's problem, using the same batching trick used in the previous section. This leads to an algorithm with a running time of $O(n^{2-1/2^{2d+9}}\log n)$. We omit the details.

Theorem 5. *In $O(n^{2-1/2^{2d+9}}\log n)$ time it is possible to compute the threshold time of the steady-state convex hull of n points moving in the plane according to polynomial functions of time of maximum degree d.*

We close this section with a few remarks about Atallah's problem. Our technique clearly is general enough to be applied to other problems (e.g., closest/farthest pairs). An

interesting question is to determine whether a more ad hoc treatment of these problems might lead to a more efficient solution. For example, a continuity argument easily shows that ensuring the local coherence of the steady-state Voronoi diagram is sufficient to compute its threshold (e.g., checking the nonzero length of its edges). It is then fairly simple to devise an $O(n \log n)$ algorithm for computing the steady-state Voronoi diagram of n moving points as well as its threshold. Note that the same argument can be made for convex hulls if all the points are guaranteed to lie on it. One essential feature of these *easy* cases is that the output contains all the input. Is this in general a necessary condition of efficiency?

7. A Discussion of the Batching Technique

In the preceding sections we have given a few examples of techniques for obtaining (slightly) subquadratic solutions to a large variety of geometric problems which admit trivial exhaustive quadratic solutions. Many other problems yield to our technique. For example,

(i) Given a set of m red objects (algebraic curves, surface patches, etc.) and n blue objects, does any red object intersect any blue object? Hopcroft's problem, mentioned above, is such a problem; detecting intersection beween a collection of red segments and a collection of blue segments is another example.

(ii) Given m rays and n triangles in 3-space, find the first triangle hit by each of the rays, or alternatively, find the number of triangles *stabbed* by each ray.

(iii) Given a collection of n (disjoint) triangles in three dimensions, find all pairs of mutually visible vertices. Here, we regard each pair of vertices as a query point z in E^6; each triangle Δ corresponds to a Boolean predicate that expresses the fact that z is not blocked by Δ. Batching the triangles and using our technique, we can obtain a subcubic solution (a cubic solution being trivial).

Reflecting on our results, we can see a whole spectrum of problems amenable to our techniques.

(A) The simplest of them admit linear or near-linear data structures and can be solved in $O(n \log n)$ time, e.g., the closest-pair problem in the plane.

(B) Next in line, we have problems for which we have efficient, polynomial-size data structures and for which the batching technique is very effective, e.g., computing the diameter of n points in E^3, Hopcroft's problem (section 5).

(C) Then we have the problems of the type discussed in this paper: sufficiently complicated as to offer no obvious alternatives but computing Collins decompositions (in theory, that is).

(D) Finally, we have problems for which it is not clear even how to obtain a Collins decomposition of the form used in this paper. For example, consider a variant of problem (iii) above in which we want all pairs of mutually visible triangles (i.e., pairs where a point of one triangle can see some point of the other one.) Can this problem be solved in subcubic time using our technique? As it turns out, the best solution currently known runs in time $O(n^4\alpha(n))$ (McKenna and O'Rourke, 1988).

A final comment concerns the use of probabilistic algorithms for the problems studied in this paper. Such a method has been developed recently in (Haussler and Welzl, 1987, Clarkson 1987, 1988, Edelsbrunner, Guibas, and Sharir, 1988). It can be roughly described as a divide-and-conquer paradigm which uses random sampling of a small subset of the input objects to obtain a cell decomposition such that, with high probability, each cell contains (or intersects) only a small number of the given objects. If the queries are known in advance (which is the case in the problems studied in section 5) then we can partition them among the cells of the decomposition, so that each point can interact only with a small number of objects. We believe that one can develop a general framework for blending this randomized method with our Collins decomposition technique, and obtain (slightly) improved probabilistic algorithms for such problems. We leave this as an open problem.

Acknowledgments

We wish to thank Johan Swenker for providing us with useful comments about this work. We also thank the referees for their diligence and scrutiny, which helped improve the presentation of this paper.

Bernard Chazelle wishes to acknowledge the National Science Foundation for supporting this research in part under Grant CCR–8700917. Micha Sharir is pleased to acknowledge the support of the Office of Naval Research under Grant N00014–87–K–0129, the National Science Foundation under Grant DCR–83–20085, the Digital Equipment Corporation, the IBM Corporation, and the NCRD — the Israeli Council on Research and Development.

References

Aho, A.V., Hopcroft, J.E., Ullman, J.D. (1974). *The design and analysis of computer algorithms*, Reading, MA, Addison-Wesley.

Arnon, D.S. (1981). *Algorithms for the geometry of semi-algebraic sets*, Tech. Rep. 436, Computer Science Dept., University of Wisconsin, Madison.

Arnon, D.S., Collins, G.E., McCallum, S. (1984a). *Cylindrical algebraic decomposition I: the basic algorithm*, SIAM J. Comput., 13, 865–877.

Arnon, D.S., Collins, G.E., McCallum, S. (1984b). *Cylindrical algebraic decomposition II: an adjacency algorithm for the plane*, SIAM J. Comput., 13, 878–889.

Atallah, M.J. (1985). *Dynamic computational geometry*, Comput. Math. with Applications, 11, 1171–1181.

Ben-Or, M., Kozen, D., Reif, J. (1984). *The complexity of elementary algebra and geometry*, Proc. 16th Ann. ACM Symp. on Theory of Computing, 457–464.

Brown, W., Traub, J.F. (1971). *On Euclid's algorithm and the theory of subresultants*, J. ACM, 18, 505–514.

Chazelle, B. (1982). *A theorem on polygon cutting with applications*, Proc. of 23rd Ann. IEEE Symp. on Foundat. of Computer Science, 339–349.

Chazelle, B. (1987). *Some techniques for geometric searching with implicit set representations*, Acta Informatica, 24, 565–582.

Chazelle, B., Guibas, L.J. (1989). *Visibility and intersection problems in plane geometry*, Disc. and Comput. Geom., 4, 551–581.

Clarkson, K.L. (1987). *New applications of random sampling in computational geometry*, Disc. Comp. Geom., 2, 195–222.

Clarkson, K. (1988). *Applications of random sampling in computational geometry, II*, Proc. 4th Ann. ACM Sympos. Comput. Geom., 1–11.

Cohen, P.J. (1969). *Decision procedures for real and p-adic fields*, Comm. Pure and Applied Math., 22, 131–151.

Cole, R. (1986). *Searching and storing similar lists*, J. Algorithms, 7, 111–119.

Collins, G.E., (1975). *Quantifier elimination for real closed fields by cylindric algebraic decomposition*, Proc. 2nd GI Conf. on Automata Theory and Formal Languages, Springer-Verlag, LNCS 33, Berlin, 134–183.

Collins, G.E., Loos, R. (1976). *Polynomial real root isolation by differentiation*, Proc. ACM Symp. on Symbolic and Algebraic Computations, Yorktown Heights, NY, 15–25.

Dobkin, D.P., Lipton, R.J. (1976). *Multidimensional searching problems*, SIAM J. Comput. 5, 181–186.

Dobkin, D.P., Silver, D. (1988). *Recipes for Geometry and Numerical Analysis - Part I: An Empirical Study*, Proc. 4th Ann. ACM Sympos. Comput. Geom., 93–105.

Edelsbrunner, H., Guibas, L.J., Stolfi, J. (1986). *Optimal point location in a monotone subdivision*, SIAM J. Comput., 15, 317–340.

Edelsbrunner, H., Guibas, L.J., Sharir, M. (1988). *The complexity of many faces in arrangements of lines and of segments*, Proc. 4th Ann. ACM Sympos. Comput. Geom., 44–55.

ElGindy, H., Avis, D. (1981). *A linear algorithm for computing the visibility polygon from a point*, J. of Algorithms, 2, 186–197.

Ghosh, S.K., Mount, D.D. (1987). *An output sensitive algorithm for computing visibility graphs*, Proc. 28th Annu. IEEE Symp. on Foundat. of Comput. Sci., 11–19.

Haussler, D., Welzl, E. (1987). *Epsilon-nets and simplex range queries*, Disc. Comp. Geom. 2, 127–151.

Hershberger, J. (1987). *Finding the visibility graph of a simple polygon in time proportional to its size*, Proc. 3rd Ann. ACM Sympos. Comput. Geom., 11-20.

Hoffmann, C.M., Hopcroft, J.E., Karasick, M.S. (1988). *Towards Implementing Robust Geometric Computations*, Proc. 4th Ann. ACM Sympos. Comput. Geom., 106–117.

Kapoor, S., Maheshwari, S.N. (1988). *Efficient algorithms for Euclidean shortest path and visibility problems with polygonal obstacles*, Proc. 4th Ann. ACM Sympos. Comput. Geom., 172–182.

Mahler, K. (1964). *An inequality for the discriminant of a polynomial*, Michigan Math. J., 11, 257–262.

McKenna, M. (1986). *The biggest stick problem*, First Computational Geometry Day, New York University.

McKenna, M., O'Rourke, J. (1988). *Arrangements of lines in 3-space: a data structure with applications*, Proc. 4th Ann. ACM Sympos. Comput. Geom., 371–380.

Milnor, J. (1964). *On the Betti numbers of real varieties*, Proc. Amer. Math. Soc. 15, 275–280.

Monk, L. (1974). *Elementary recursive decision procedures*, PhD thesis, University of California, Berkeley.

Preparata, F.P., Shamos, M.I. (1985). *Computational geometry: an introduction*, Springer-Verlag, New York, NY.

Sarnak, N., Tarjan, R.E. (1986). *Planar point location using persistent search trees*, Comm. ACM 29, 669–679.

Seidenberg, A. (1954). *A new decision method for elementary algebra*, Annals of Math., 60, 365–374.

Schwartz, J.T., Sharir, M. (1983). *On the "piano movers" problem. II: General techniques for computing topological properties of real algebraic manifolds*, Adv. in Appl. Math, 4, 298–351.

Tarski, A. (1948). *A decision method for elementary algebra and geometry*, Univ. of Calif. Press, 1948, 2nd edition, 1951.

van der Waerden, B.L. (1953). *Modern Algebra*, Ungar Co., New York.

Yao, A.C. (1982). *On constructing minimum spanning tree in k-dimensional space and related problems*, SIAM J. Comput., 11, 721–736.

Yao, A.C., Yao, F.F. (1985). *A general approach to d-dimensional geometric queries*, Proc. 17th Ann. ACM Symp. on Theory of Computing, 163–168.