Computational Geometry — A Survey

D. T. LEE, SENIOR MEMBER, IEEE, AND FRANCO P. PREPARATA, FELLOW, IEEE

Abstract — **We survey the state of the art of computational geometry, a discipline that deals with the complexity of geometric problems within the framework of the analysis of algorithms. This newly emerged area of activities has found numerous applications in various other disciplines, such as computer-aided design, computer graphics, operations research, pattern recognition, robotics, and statistics. Five major problem areas — convex hulls, intersections, searching, proximity, and combinatorial optimizations — are discussed. Seven algorithmic techniques — incremental construction, plane-sweep, locus, divide-and-conquer, geometric transformation, prune-and-search, and dynamization — are each illustrated with an example. A collection of problem transformations to establish lower bounds for geometric problems in the algebraic computation/decision model is also included.**

Index Terms — **Algebraic computation tree, analysis of algorithms, combinatorial optimization, computational complexity, computational geometry, convex hull, divide and conquer, dynamization, geometric transformation, plane sweep, proximity.**

I. Introduction

COMPUTATIONAL geometry, as it stands nowadays, is concerned with the computational complexity of geometric problems within the framework of analysis of algorithms. The phrase, however, has been used in at least two other connotations. Bezier [47], Forrest [146], and Riesenfeld [299] have studied geometric modeling by means of spline curves and surfaces, a topic that is closer to numerical analysis than it is to geometry, and Forrest refers to this discipline as "computational geometry." In their book entitled *Perceptrons* (of which the subtitle is also *Computational Geometry*), Minsky and Papert [264] deal with the complexity of predicates that recognize certain geometric properties, such as convexity. The intent of their work was to identify the capabilities of large retinas composed of simple circuits to perform pattern recognition tasks.

We shall concentrate on the prevailing connotation of computational geometry, which has now become a discipline by itself in algorithm design and analysis. A large number of applications areas such as pattern recognition [328], computer graphics [268], image processing [286], operations research, statistics [41], [314], computer-aided design, robotics, etc., have been the incubation bed of the discipline since they provide inherently geometric problems for which efficient algorithms have to be developed. These problems include the Euclidean traveling salesman, minimum spanning tree, linear programming, and hosts of others. Algorithmic studies of these and other problems have appeared in the scientific literature with an increasing intensity in the past two decades and a growing number of researchers have been attracted to this discipline, christened "Computational Geometry" in a paper by Shamos [313] in 1975. Because the research results on the subject are spread in the literature, it is appropriate at this time to survey in some detail this discipline. The bibliography of Edelsbrunner and van Leeuwen [139] is a good source for locating most of the articles related to this field up to the Summer of 1982; a more complete and detailed digest of computational geometry is due to appear as a textbook in the near future [295].

According to the nature of the geometric objects involved, we can identify basically five categories into which the entire collection of geometric problems can be conveniently classified, i.e., convexity, intersection, geometric searching, proximity, and optimization. As far as problem solving techniques developed to date, we can identify seven major paradigms, i.e., incremental construction, plane-sweep, locus, divide-and-conquer, geometric transformations, prune-and-search, and dynamization. We shall briefly illustrate each of these paradigms by an example. Before proceeding, however, it is appropriate to carefully discuss the model of computation and the measures of complexity.

Models of Computation: The usually adopted computational model is a random access machine (RAM), similar to that described in [1] with the added feature of real number arithmetics. That is, in this machine each memory location can hold a real number and each arithmetic operation, such as addition, multiplication, and division, can be performed in unit time. Depending on the problems at hand, we have various other primitive operations, like computing the intersection of two straight lines, or computing the distance between two points. These primitive operations are all assumed to take constant time to execute. Basically, we can categorize geometric problems into two classes, conventionally called "computation" and "decision." A *computation problem* requires the construction of some geometric object satisfying a given property, whereas a *decision problem* tests whether a geometric object possesses or not a given property. For any instance of computation problem we can almost always transform it into a corresponding instance of decision problem, and a lower bound for the decision problem can then be used

Manuscript received March 1, 1984; revised July 1, 1984. This work was supported in part by the National Science Foundation under Grants MCS 83-42682 and MCS 81-05552.

D. T. Lee is with the Department of Electrical Engineering and Computer Science, Northwestern University, Evanston, IL 60201.

F. P. Preparata is with the Coordinated Science Laboratory, University of Illinois, Urbana, IL 61801.

0018-9340/84/1200-1072$01.00

to establish a lower bound for the computation problem. (See Section III-F for more details about problem transformations and lower bounds.) Thus, we may restrict ourselves to the appropriate computation model for decision problems: this is the so-called algebraic decision tree [26], [108], [296], [298] model of computation. This model of computation will be adopted throughout, unless otherwise specified, in order to establish a lower bound on the computation time for decision problems and their associated computation problems.

An *algebraic computation tree* [26] on a set of variables $V = \{x_1, x_2, \cdots, x_n\}$ where $x_i \in \boldsymbol{R}$, is a binary tree T with a function that assigns the following.

1) To any node v with exactly one son (simple node) an operational instruction of the form $f_v := f_{v_1} \# f_{v_2}$, or $f_v := c \# f_{v_1}$, or $f_v := \sqrt{f_{v_1}}$ where v_i $(i = 1, 2)$ is an ancestor of v in T, or $f_{v_i} \in V$, $\# \in \{+, -, \times, /\}$, and $c \in \boldsymbol{R}$ is a constant.

2) To any node v with two sons (branching node) a comparison instruction of the form $f_{v1} > 0$, or $f_{v1} < 0$, or $f_{v1} = 0$ where $v1$ is an ancestor of v in T or $f_{v1} \in V$.

3) To any leaf an output YES or NO.

Let $W \subseteq \boldsymbol{R}^n$ be any set. The *membership problem* (a decision problem) for W is to decide if point $x = (x_1, x_2, \cdots, x_n) \in \boldsymbol{R}^n$ belongs to W. Given any x, the program traverses a path $P(x)$ in T starting from the root. At each simple node the associated arithmetic operation is executed, and at each branching node a branching is made according to the outcome of the comparison at the node. When a leaf is reached, the answer YES or NO is returned. It is assumed that whenever a node v is encountered and its associated operation is a division, the denominator is nonzero, and that when it is a square root operation, the operand is nonnegative. The computation tree T is said to **solve** the membership problem if the answer returned is correct for every input $x \in \boldsymbol{R}^n$. The complexity of T, denoted $C(T)$, is defined to be the maximum of cost (x, T) for any x where cost (x, T) is the number of nodes traversed in the path $P(x)$ in T. The complexity of the decision problem for W, denoted $C(W)$, is the minimum of $C(T)$ for any algebraic computation tree T that solves the membership problem.

A different formulation of the algebraic computation tree is the so-called dth order (algebraic) decision tree [324] for determining if $x \in W \subseteq \boldsymbol{R}^n$. A dth order decision tree is one in which each node of the tree has the form of a comparison $f(x)\ ?\ 0$ where f is a polynomial of the input of degree at most d and $? \in \{<, >, =\}$. When d equals 1, the decision tree becomes a **linear decision tree**, on which several lower bound proofs are based [99], [107], [108], [298], [349], [352].

Time and Space Complexities: The time and space used by an algorithm are two major measures for the efficiency of the algorithm. Normally, we count only the number of key operations, such as comparisons, performed by the algorithm and express it as a function of input size. In doing so, we must ensure that the number of unaccounted-for operations is proportional to that of key operations so that the running time of the algorithm is within a constant factor of the estimated one. As for the space requirement, we count the maximum amount of storage ever needed during the execution of the algorithm. This, too, is expressed as a function of input size. The standard notation has been suggested by Knuth [204], and is given below.

$O(f(n))$ denotes the set of all functions $g(n)$ such that there exist positive constants C and n_0 with $|g(n)| \leq Cf(n)$ for all $n \geq n_0$;

$\Omega(f(n))$ denotes the set of all functions $g(n)$ such that there exist positive constants C and n_0 with $g(n) \geq Cf(n)$ for all $n \geq n_0$;

$\theta(f(n))$ denotes the set of all functions $g(n)$ such that there exist positive constants C, C', and n_0 with $Cf(n) \leq g(n) \leq C'f(n)$ for all $n \geq n_0$;

$o(f(n))$ denotes the set of all functions $g(n)$ such that *for all constants* C there is an n_0 with $g(n) \leq Cf(n)$ for all $n \geq n_0$.

There are two types of analyses: worst case and expected case. For the worst case analysis we seek the maximum amount of time/space used by the algorithm for all possible inputs. For the expected case analysis we normally assume a certain probabilistic distribution on the input and study the performance of the algorithm for any input drawn from the distribution. Mostly, we are interested in the **asymptotic analysis,** i.e., the behavior of the algorithm as the input size approaches infinity. Since expected case analysis is usually harder to tackle, and moreover the probabilistic assumption sometimes is difficult to justify, emphasis will be placed on the worst case analysis. Unless otherwise specified, we shall consider only worst case analyses.

II. Problem Solving Techniques

We now give an example for each of the seven major problem solving techniques mentioned above.

A. Incremental Construction

This is the simplest and most intuitive problem solving technique and is also known as the **iterative greedy method**. The main idea is that we construct or compute the solution in an iterative manner. An analog to this technique that is better known is that of sorting-by-insertion [203], in which the sorted list is obtained by inserting one by one the elements into the partial sorted list. Specific examples follow.

Consider the problem of computing the line arrangements in the plane. That is, given a set H of n straight lines in the plane, compute the partition of the plane induced by H. The obvious approach is to construct the partition by considering one line at a time and building up the partition iteratively [86], [133]. In Fig. 1, when line i is inserted, we need to traverse the regions that are cut by the line and construct the new partition at the same time. As it turns out, the approach takes $O(n^2)$ time and is asymptotically optimal in the sense that the total running time is proportional to the space required to represent the partition. Furthermore, the approach generalizes to higher dimensions [133]. Since this technique is self-explanatory, we omit the description of the algorithm and state the result as a theorem.

Theorem 1: The problem of computing the arrangement of n lines in the plane can be solved in $O(n^2)$ time by the incremental construction technique.

As another example for which the incremental approach

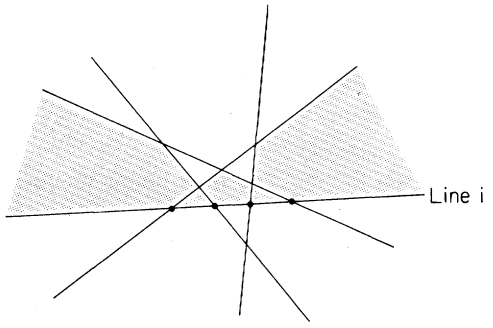

Fig. 1. Construction of line arrangement.

works well, and in fact can be shown to be optimal, consider the problem of finding the convex hull of a set of points in the plane (cf. Section II-D). We simply consider the points one at a time and construct the convex hull of the points on the fly. If the next point lies in the convex hull just constructed, it is ignored. Otherwise, it is an extreme point and the convex hull must be updated by computing the two extreme supporting lines anchored at the new point. See [21], [221], [291] for an optimal implementation of this approach. Although these two examples illustrate that the incremental construction technique yields optimal solutions, it is normally not the case for the majority of the problems considered in the literature.

B. Plane Sweep

As the term suggests, the scheme is primarily suited for 2-dimensional problems, but its generalization to higher dimensions is straightforward. See, for example, [50]. This technique is also know as the **scan-line method** in computer graphics [268], and is used for a variety of applications, such as shading, polygon filling, among others [58], [215], [286]. For ease of illustration we shall describe the scheme in connection with a specific example and indicate the main features associated with it.

Consider the problem of reporting all intersecting pairs among a set of horizontal and vertical line segments. Let S denote the set of line segments, $S = \{s_1, s_2, \cdots, s_n\}$, such that s_i is either horizontal or vertical; if s_i is horizontal, it is specified by $(x_{i1}, x_{i2}; y_i)$, and if s_i is vertical, it is specified by $(x_i; y_{i1}, y_{i2})$ where x's and y's are x- and y-coordinates of the endpoints of each segment. (For simplicity, no two segments are collinear.) Imagine we "sweep" a vertical line across the plane from left to right (or a horizontal line from top to bottom) and report an intersecting pair of segments as we reach their common point in the sweep. It is obvious that no new intersecting pairs will be found between the x-coordinates of endpoints of segments. In other words, those abscissas are special positions at which intersections may occur and they are called "event points" [269], [295]. Observe also that a sweep line at abscissa x partitions the set of segments into three subsets, S_1, S_2, and S_3 where S_1 is the set of segments lying entirely to the left of the sweep line, S_2 is the set of segments intersecting the sweep line, and S_3 the set of segments lying entirely to the right of the sweep line. The set S_1 of segments will not play any role in the "future" of the left-to-right sweep, i.e., for sweep lines at $x' > x$; the set S_2 contains the "active" segments, which may be part of the output, i.e., may intersect at a later time a segment in S_3; and S_3 is the set which has not played any role up to this point. The key of this technique is to maintain "relevant" information at each event point about the active segments in S_2. As soon as a segment becomes active, it is kept as part of the relevant information, and when it is past the sweep line, it is deleted.

For our example the relevant information is just the sequence of the ordinates of all active segments, stored in a height-balanced tree [1]. If the next "event" corresponds to a vertical segment, this segment is used to query the "status" data structure and all horizontal segments whose y's lie in the interval defined by the vertical query segment are reported to intersect it. If the next event corresponds to the left endpoint of a horizontal segment, the horizontal segment becomes active and is inserted into the tree; otherwise, the event is the right endpoint of a horizontal segment, which is therefore to be deleted from the tree. The correctness of this algorithm can be easily established. From the performance viewpoint since at each event point we need to perform an insertion, or a deletion, or a report, each requiring $O(\log n)$ time (plus output in case of reporting), the total time is $O(n \log n + k)$ where k is the number of intersecting pairs reported. We have the following theorem.

Theorem 2: The problem of reporting all k intersecting pairs of line segments among a set of n horizontal and vertical line segments can be solved in $O(n \log n + k)$ time by the plane-sweep technique.

In summary, there are two basic data structures associated with this plane-sweep technique, i.e., i) the *event point schedule,* which is a sequence of abscissas, ordered from left to right, and ii) the *sweep line status,* which is an appropriate description of the relevant information of the geometric objects at the sweep line. Note that the data structures may be different under various situations and that the event point schedule may be dynamically updated during the execution of the algorithm. For the example discussed above, the event point schedule is a fixed ordered list, and the sweep line status is realized as a height-balanced tree. Plane-sweep applications for which the event point schedule is dynamically changing can be found in [38], [63], [229], [269]; in this case, a priority queue [1] of some kind is normally used.

C. Locus

This method is mostly associated with geometric searching problems in the so-called repetitive mode (an arbitrarily long sequence of queries on a fixed file), in which queries of a given kind are to be handled efficiently [278]. For example, we want to preprocess a set S of points in the plane so that a query, known as *range searching query,* calling for the report of points in S that lie in the interior of a rectangular window, can be answered quickly. Using this method, we would like to partition the query space into "cells" such that all points in a given cell generate the same response to a query. More formally, we partition the space into a number of "equivalence" classes, whose underlying relation is dependent on the problem itself. Consider, for example, the *2-dimensional dominance problem,* which is given below.

2-Dimensional Dominance Problem: Given a set S of n points, $p_1, p_2, \cdots, p_n$, in the plane, with preprocessing allowed, find the *number* of points in S *dominated* by a given point q. (A point p is said to be *dominated* by a point q if both x- and y-coordinates of p are no greater than those of q, respectively.)

We show how the locus method enables one to answer the query in $\theta(\log n)$ time, which is optimal. First we define a relation T in $\boldsymbol{R}^2$ such that (p, q) in $\boldsymbol{R}^2$ is in T if the subsets of S dominated by p and by q, respectively, are identical. The relation T, being an equivalence relation, will induce a partition of the plane into, in general, $(n+1)^2$ equivalence classes. Geometrically, if we draw vertical and horizontal lines through each point in S, these n vertical and horizontal lines will partition the plane into $(n+1)^2$ cells. Each cell is an equivalence class since for any two points in a cell the subsets of S dominated by these two points are identical. Thus, the problem becomes an instance of *point-location problem*—locate the query point in the cells—and the cell in which the query point lies will contain the solution to the problem (an integer). Since two binary searches suffice to locate any query point in the $O(n^2)$ cells defined above, we have the following theorem.

Theorem 3: The 2-dimensional dominance problem can be solved in $O(\log n)$ time with $\theta(n^2)$ preprocessing time and space using the locus method.

Other problems for which the locus method is applicable are nearest neighbor searching [147], [157], [354], shortest path finding in the presence of obstacles [229], [327], etc.

D. Divide-and-Conquer

This is a classical problem solving technique and has proven its value for geometric problems as well [30], [40], [42], [83], [172], [174], [198], [205], [220], [228], [293], [315]. This technique normally involves partitioning of the original problem into several subproblems, recursively solving each subproblem, and then combining the solutions to the subproblems to obtain the solution to the original problem. A well known example for which the technique works is the **convex hull** problem. That is, given a set S of n points in the plane, find the convex hull of S (the smallest convex set containing S). A divide-and-conquer algorithm for computing the convex hull of S is given below. Here we assume that the input is just a collection of points specified by their x- and y-coordinates, and the output is a sequence of points on the convex hull. Note that the convex hull of a planar point set is just a convex polygon. So the output is a sequence of vertices, say, in clockwise order, of the convex polygon.

Algorithm Convex Hull (S):

If $|S| \leq 2$ **then return** (S)
else begin
  divide S into S_1 and S_2 such that $|S_1| = \lfloor 1/2 \rfloor |S|$ and $S_1 \cup S_2 = S$;
  $S' :=$ CONVEX HULL (S_1);
  $S'' :=$ CONVEX HULL (S_2);
  $T :=$ MERGE(S', S'');
  return (T)
end.

where MERGE(S', S''), the "conquer" step, is a procedure that combines two convex polygons into one, i.e., computes the convex hull of their union. It has been shown [293], [313], [316] that the union can be found in time $O(|S'|+|S''|)$ (see Problem 3.1.2 in Section III). The analysis of the divide-and-conquer algorithm is relatively easy. Let $T(n)$ denote the time for the algorithm CONVEX HULL where $n = |S|$. Then assuming that n is a power of 2, we have the following recurrence relation:

$$T(1) = \text{constant}$$

$$T(n) = 2T(n/2) + M(n/2, n/2)$$

where $M(s, t)$ denotes the time for computing the convex hull of the union of two convex polygons with s and t vertices, respectively. Since $M(n/2, n/2) = O(n)$, then $T(n) = O(n \log n)$. Thus, we have the following theorem.

Theorem 4: The convex hull problem for a set of n points in the plane can be solved in $O(n \log n)$ time by the divide-and-conquer technique.

E. Geometric Transformations

The application of a convenient transformation to the geometric objects can be frequently resorted to in oder to transform a given problem into an equivalent problem. Sometimes a transformation in the d-dimensional space maps points to points, and in other cases it maps k-dimensional varieties to $(d\text{-}1\text{-}k)$-dimensional varieties. It is an important class of the latter type—known as *polarity* or *duality*—which has been recently used very successfully in several applications. The basic reason for this success is that the transformed problem appeals more directly to intuition and is therefore more suggestive of efficient algorithms.

We shall give a specific example illustrating this point. Consider the intersection problem of n half-planes in the plane. That is, given n half-planes, each specified as a linear constraint of the form $y \leq a_i x + b_i$ or $y \geq a_i x + b_i$, find their intersection. Since the intersection is a convex polygon, which may or may not be bounded, the problem can be solved by a divide-and-conquer algorithm, in which the conquer step is the problem of finding the intersection of two convex polygons. Since intersecting two convex polygons can be done in linear time, the divide-and-conquer algorithm runs in $O(n \log n)$ time [313], [317]. However, we shall use a geometric transformation to solve this problem within the same time bound, for the bound is optimal [315] to within a constant factor under a model of computation that allows analytic functions of the input [155] or under the algebraic computation tree model of Ben-Or [26].

First of all, let us assume that no lines of the input that define half-planes are vertical. (This can always be done by rotating the coordinate axes.) Let us call a half-plane a *lower half-plane* if its boundary line is above the open half-plane and an *upper half-plane* otherwise. Thus, a lower half-plane is of the form $y \leq a_i x + b_i$, and an upper half-plane is of the form $y \geq a_i x + b_i$.

The geometric transformation employed is a point-to-line mapping, known as *polarity*. A polarity in the plane (and,

with obvious generalization, in any number of dimensions) makes reference to a conic, a second-degree curve. This conic can be chosen in a number of convenient ways. If, for example, we choose the conic as the parabola $y = x^2/2$, then point (a, b) is mapped to line $y = ax - b$ and *vice versa* (polarity is always involutory). If (a, b) is external to the parabola, its image, called *dual* or *polar,* is the line passing by the points of contact of the supporting lines from (a, b) to the parabola. This transformation preserves incidence, i.e., if point (c, d) is on line $y = ex - f$, then the same holds also for their duals: point (e, f) is on line $y = cx - d$.

Consider now the problem of finding the intersection of the lower half-planes. Note that a lower half-plane k is *redundant* if and only if there exist two lower half-planes i and j such that i) the boundary line L_k is above the point p where the boundary lines L_i and L_j of half-planes i and j intersect, and ii) the slope of L_k is between the slopes of lines L_i and L_j. In the dual plane there is a corresponding statement. Note that we transform the lines that define half-planes into points in the dual plane. A point p_k, the transform of line L_k, in the dual plane is redundant if and only if there exist two points p_i and p_j such that i) p_k is below the line L_p which is the transform of the intersection of lines L_i and L_j, and ii) the x-coordinate of p_k is between those of points p_i and p_j. In other words, a point in the dual plane is redundant if and only if it is directly below a line segment determined by two points. With this it can be shown [61] that the nonredundant upper half-planes correspond to those points on the bottom chain of the convex hull of the points in the dual plane. Since the convex hull can be computed in $O(n \log n)$ time, the intersection of n upper half-planes can be found in $O(n \log n)$ time. Similarly, the intersection of n lower half-planes can also be found in $O(n \log n)$ time. Once the intersections of upper half-planes and lower half-planes are found, the intersection of the two unbounded polygons can be found in linear time by, say, the plane sweep technique since the edges that define the intersection are ordered. Thus, we have the following theorem.

Theorem 5: The intersection of n half-planes can be found in $O(n \log n)$ time by a geometric transformation.

Note that this transformation technique can be easily extended to higher dimensions. In particular, the intersection of n upper half-spaces can be found in $O(n \log n)$ time by computing the bottom portion of the convex hull of the points in the dual space using the algorithm due to Preparata and Hong [293]. General schemes for finding the intersection of n arbitrary half-spaces in $O(n \log n)$ time can be found in [294] (see also [117]). For details and applications of geometric transformations see, for example, [61], [62], [71], [86], [132], [133], [265], [295].

F. Prune-and-Search

This approach, used by Megiddo [257]–[259] and Dyer [118], is primarily used to solve optimization problems and has shown its power in yielding efficient algorithms for a number of geometric optimization problems, one of which is the well-known linear programming problem. A brilliant polynomial time algorithm, developed some time ago by Khachian [195], is mainly of theoretic interest since it cannot compete with the more practical simplex method [96]. More recently, Megiddo and Dyer, independently, proposed a novel multidimensional search technique, that we classify as *prune-and-search* here, to obtain a linear time algorithm for the linear programming problem when the dimension is fixed [257], [259]. Note, however, that the time complexity is double-exponential in the number of dimensions. With this technique several geometric optimization problems can be solved efficiently. Specifically we quote: the linear separability problem, i.e., given n points in $\boldsymbol{R}^d$ organized into two disjoint sets, find a hyperplane, if it exists, that separates these two sets; the Chebyshev regression problem, i.e., given n points $p_i = (x_{i1}, x_{i2}, \cdots, x_{id}) \in \boldsymbol{R}^d$, $i = 1, 2, \cdots, n$, find a linear function $f(x_1, x_2, \cdots, x_d) = \sum_{j=1}^{d-1} a_j x_j + a_d$ minimizing $\max\{|\sum_{j=1}^{d-1} a_j x_{ij} + a_d - x_{id}|,\ i = 1, 2, \cdots, n\}$; the 1-center problem, i.e., given n points in $\boldsymbol{R}^d$, find the smallest hypersphere enclosing these n points. All these problems can be solved in $O(n)$ time when d is fixed. The technique can be extended to solve optimization quadratic programming problems. The details of the extension and other related problems solvable by the technique, or similar ones, can be found in [257].

We shall now give a brief sketch of this remarkable technique in two dimensions. The linear programming problem is formulated as follows:

$$\begin{aligned} &\text{minimize} && ax + by \\ &\text{subject to} && a_i x + b_i y + c_i \le 0, \qquad i = 1, 2, \cdots, n. \end{aligned}$$

The method discards not only redundant constraints, but also those constraints that are guaranteed not to contain a vertex of the feasible region extremizing the objective function. We first apply a linear transformation to the points of the plane so that the objective function becomes equal to one of the two coordinates, say, the ordinate of the plane. At this point the problem reduces to finding the extreme value of a piecewise linear convex function of the abscissa. The key feature then is that since all we want is the identification of the extremizing abscissa, we need not explicitly construct this convex function, which remains implicitly defined by a set of linear constraints.

At first, one sets $Y = ax + by$ and $X = x$. Assuming, without loss of generality, that $b \neq 0$, we have:

$$\begin{aligned} &\text{minimize} && Y \\ &\text{subject to} && \alpha_i X + \beta_i Y + c_i \le 0, \qquad i = 1, 2, \cdots, n \end{aligned}$$

where $\alpha_i = (a_i - (a/b)b_i)$, and $\beta_i = b_i/b$. In this new form we have to compute the smallest Y of the vertices of the convex polygon P (feasible region) determined by the constraints (see Fig. 2). The n constraints are partitioned into three classes, I_0, I_-, I_+, depending upon whether β_i is zero, negative, or positive. Set I_0 determines an X-interval $[u_1, u_2]$ where the solution is to be sought, while sets I_- and I_+, respectively, define, in an implicit manner, downward- and upward-convex piecewise linear functions $F_-(X)$ and $F_+(X)$ delimiting the feasible region. Thus, the original problem is transformed to

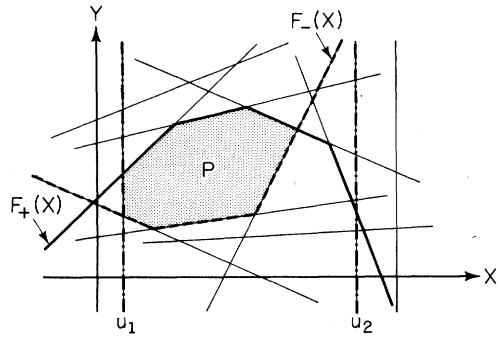

Fig. 2. Illustration of the feasible region defined by a set of linear constraints.

$$\begin{aligned} &\text{minimize} && F_-(X) \\ &\text{subject to} && F_-(X) \le F_+(X) \\ & && u_1 \le X \le u_2 . \end{aligned}$$

The values of $F_-(X)$ and $F_+(X)$ at any X can be computed in time $O(n)$, as can their slopes. Thus, within this time bound, for any $X' \in [u_1, u_2]$ one of the following conclusions can be reached: i) X' is infeasible, and there is no solution to the problem; ii) X' is infeasible and we know on which side of X' (right or left) any feasible value of X may lie; iii) X' is feasible and we know on which side of X' the minimum of $F_-(X)$ lies; and iv) X' achieves the minimum of $F_-(X)$. Cases i) and iv) represent final solutions.

To choose X' we partition both I_+ and I_- into pairs of constraints (straight lines) and determine the abscissas X_{ij} of their intersection. If $X_{ij} \notin [u_1, u_2]$, one constraint can be immediately eliminated as redundant. For the set of $X_{ij} \in [u_1, u_2]$, we find in time $O(n)$ [1], [111], [306] their median X'', and evaluate $F_+(X'')$ and $F_-(X'')$. Then, by the above argument, half of the X_{ij}'s lie in a region not containing the extremizing value, so that one constraint of the pair can be eliminated. In this manner for each evaluation of $F_+(X)$ and $F_-(X)$ a fixed fraction α of the currently retained constraints can be eliminated. This leads to the conclusion that in $\log_{1/(1-\alpha)} n$ stages the size of the set of retained constraints becomes sufficiently small to allow a direct solution. Since the time spent is $Cn + C\alpha n + C\alpha^2 n + \cdots = O(n)$, for some constant C, we have the following conclusion.

Theorem 6: The 2-dimensional linear programming problem with n constraints can be solved in $O(n)$ time.

G. Dynamization Technique (Static-to-Dynamic Transformation)

The techniques are developed for problems whose database is changing over (discrete) time. The idea is to make use of good data structures for a static (fixed) database and add to them certain dynamization mechanisms so that insertions or deletions of elements in the database can be accommodated efficiently.

As before, we shall describe the general approach by an example. The example falls in the category of **geometric searching problems.** A typical searching problem is the **membership** problem, i.e., given a set F of objects, is x a member of F? If F is the set of reals, we might be interested in the nearest neighbor of x, i.e., the element in F which is closest to x. This is known in the literature as best match [64], [156] or nearest neighbor searching [147], [157], [171], [354] problem. More formally, a general query is a question containing a variable of type $T1$ and is asked of a set of elements of type $T2$, and the answer is of type $T3$. In the membership query, $T1$ and $T2$ are the same and $T3$ is Boolean; while in the nearest neighbor query, $T1$, $T2$, and $T3$ are all identical, i.e., reals. The query Q can be viewed as a mapping from $T1$ and subsets of $T2$ to $T3$, i.e., $Q: T1 \times 2^{T2} \rightarrow T3$, [39], [304].

The class of geometric searching problems to which dynamization techniques are applicable is the so-called *decomposable searching problems,* defined below.

Definition [29]: A searching problem with query operation Q is decomposable if there exists an efficiently computable binary (associative and commutative) operator @ satisfying the condition

$$Q(x, A \cup B) = @(Q(x, A), Q(x, B)) .$$

It is easy to see that both examples given above are decomposable.

For a searching problem we have a certain data structure for the set of elements on which the searching is performed. Associated with a data structure A there are three cost measures, i.e.,

1) $P_A(N)$, the preprocessing time required to build A,
2) $Q_A(N)$, the query time required to search A, and
3) $S_A(N)$, the storage required to represent A

where N denotes the number of elements of the set represented in A.

Dynamization involves some kind of transformation that converts a static data structure into a dynamic one, allowing insertion or deletion of elements. Let us consider the problem of nearest neighbor searching in the plane. Given a set of n points in the plane, find the nearest neighbor of a query point x. We have for the static version of this problem the following performance measures: $P_A(n) = O(n \log n)$, $Q_A(n) = O(\log n)$, and $S_A(n) = O(n)$ where A is, for example, any of the data structures of Lipton and Tarjan [242], Kirkpatrick [197], or Edelsbrunner *et al.* [128]. Now let us see how we can convert the data structure A into a dynamic one, denoted D, to support insertions, deletions, and queries. There are a number of dynamization techniques [121], [164], [166], [248], [262], [263], [276], [279], [280], [282], [304], [339], [340], [341] known in the literature, but we shall describe the technique developed by van Leeuwen and Wood [341], which provides the general flavor of the approach.

The general principle is to organize the file as a collection of separate data structures, so that each update can be confined to one (or, possibly, a fixed small number) of them; however, to avoid shifting the burden to the query activity, one must refrain from excessive fragmentation since queries normally involve the entire collection. With this general idea, let $\{x_k\}_{k \ge 1}$ be a sequence of increasing integers, called *switchpoints* where x_k is divisible by k and $x_{k+1}/(k+1) > x_k/k$. Let $x_0 = 0$, $y_k = x_k/k$, and let n denote the current size

of the point set. The point set is partitioned into a collection of subsets, each organized into a separate static data structure. For a given k, called the *level,* the dynamic data structure D consists of $(k + 1)$ structures of the same type A: k of them, called *blocks,* have sizes comprised between y_k and y_{k+1}, whereas one, called *dump,* has size between 0 and $(y_{k+1} - 1)$. Each such structure B is equipped with a counter $s(B)$ determining its *status* as "low," "partial," or "full," depending upon whether $s(B) = y_k$, $y_k < s(B) < y_{k+1}$, or $s(B) = y_{k+1}$, respectively.

As mentioned earlier, the query involves all $(k + 1)$ structures, for a total query time $O(kQ_A(y_{k+1}))$ since y_{k+1} is a bound to their sizes. More delicate is the handling of updates where, to control the cost, it is necessary to control the maximum size of the data structures. Insertions are easier since they can all be made in the dump; deletions may occur anywhere, and it may be necessary to follow up a deletion in a block with a transfer from another block (specifically, from a partial block if the deletion occurred in a low block). Of course, the static data structures must be supplemented by a dictionary [1] of size n to identify in time $C(n)$ the block where the deletion is to occur. In all cases, the cost of an update is $O(U_A(y_{k+1}) + C(n))$ since y_{k+1} is the maximum size of the static structures. We now note that large variations of n are accompanied by more moderate variations of the parameter k. Indeed, when all blocks are full at level k, we switch to level $k + 1$ by the time the dump size also reaches y_{k+1}. This is done by including the dump in the collection of blocks (making for a total of $k + 1$ blocks), renaming them from "full" status to "low" status and initializing a new dump with 0 elements. Notice that when we switch from level k to level $k + 1$, we have exactly $(k + 1)y_{k+1} = x_{k+1}$ points in the set. On the other hand, when the set size goes from x_k to $x_k - 1$, we switch from level k to level $k - 1$ by an analogous process where a block is degraded to be the dump. It can be shown [341] that level switching can be correctly accomplished in time $O(1)$. We summarize the above discussion in the following theorem.

Theorem 7: Any static data structure A used for a decomposable searching problem can be transformed into a dynamic data structure D for the same problem with the following performance. If $x_k \leq n < x_{k+1}$, the query time $Q_D(n)$ for D is $Q_D(n) = O(kQ_A(y_{k+1}))$. The update time $U_D(n)$ (deletion or insertion) for D is $U_D(n) = O(C(n) + U_A(y_{k+1}))$. The space required is $S_D(n) = O(kS_A(y_{k+1}))$.

If we choose, for example, the switchpoints x_k to be the first multiple of k that is greater than or equal to 2^k, then k is about $\log_2 n$, and y_k is about $n/\log_2 n$. Since there exists a data structure A for nearest neighbor searching with $Q_A(n) = O(\log n)$ and $U_A(n) = P_A(n) = O(n \log n)$ [197], [242], we have the following corollary.

Corollary 1: The nearest neighbor searching problem in the plane can be solved in $O(\log^2 n)$ query time and $O(n)$ update time. (Note that $C(n)$ in our case is $O(\log n)$.)

There are several other dynamization schemes that deal with various query-time/space and query-time/update-time tradeoffs. For instance, in the current scheme if a different sequence of switchpoints were chosen, different query time and update time would result. The interested reader is referred to the aforementioned references for details.

III. Classes of Problems

In this section we shall survey each of the problem classes we have mentioned in the Introduction.

A. Convex Hulls

The problem of the convex hull is not only central to practical applications, but is also the vehicle for the solution of a number of other significant questions in computational geometry.

Problem 3.1.1 (Convex hull): Given a set S of n points in d-dimensional space, find its convex hull. (The convex hull of a set is the smallest convex set that contains the set; the boundary of the convex hull of S is denoted $CH(S)$.)

The convex hull problem, particularly for a set of points in the plane, has been studied extensively and has applications in pattern recognition [5], [115], image processing [301], and stock cutting and allocation [152]. We have seen in Section II-D that this problem can be solved by divide-and-conquer in $O(n \log n)$ time. There is a long list of articles containing results on the convex hull of a planar point set [3], [4], [6], [7], [31], [32], [42], [66], [68], [120], [167], [184], [293], [310], [313]. Their running times are either $O(n \log n)$ or $O(nH)$ where H is the number of points on the convex hull, except for [68] and [310], which deal with higher dimensional convex polytopes.

We remark here that the output of the planar convex hull problem is an *ordered* list of vertices on the convex hull. Therefore, $O(n \log n)$ time is both sufficient and necessary (cf. Lemma 3.6.2).

We begin by briefly reviewing two early approaches to the solution of this problem, the Graham's scan [167] and the Jarvis' march [184]. These two schemes contain many seminal ideas for this topic. See [333], in which credit is attributed to Bass and Schubert [25] for giving the first convex hull finding algorithm.

Graham's Scan: Graham, in one of the first papers dealing with geometric problems [167], presented an $O(n \log n)$ algorithm to compute the convex hull of n points in the plane. The algorithm works as follows. First, an internal point O is chosen arbitrarily, e.g., the centroid of three noncollinear given points. Second, the data points are sorted in angular order about the point O. Then a point is selected that is known to be on the convex hull, e.g., the point v_0 with the minimum y-coordinate (and maximum x-coordinate, if there are ties). Suppose that the points are maintained as a sorted list $v_0, v_1, \cdots, v_{n-1}$, counterclockwise around the origin, such that $v_i = \text{NEXT}(v_{i-1}), i = 1, 2, \cdots, n$ and $v_n = v_0$. Pointer PREV gives the reverse clockwise order. We now perform a counterclockwise scan of the vertices, taking three points at a time. We say that the three points t, u, v form a *left turn,* if v lies strictly to the left of the directed line from t to u. The scan works as follows.

```
begin v := v0;
    while NEXT(v) ≠ v0 do
        if v, NEXT(v), NEXT(NEXT(v)) form a left turn
        then v := NEXT(v)
        else begin
                DELETE NEXT(v);
                if v ≠ v0 then v := PREV(v)
             end
end
```

That is, whenever a left turn is encountered, we advance; otherwise, the middle point NEXT(v) is not on the convex hull and should be deleted (and never reexamined). When a deletion occurs, we *backtrack* to verify if v is still a hull vertex. So we either advance in the list or backtrack on the hull. Since each step takes a constant time, the scan is completed in linear time. The entire algorithm therefore runs in $O(n \log n)$ time, the sorting step being dominant.

Jarvis' March: Observing that, independent of the final convex hull, the Graham's algorithm always takes $O(n \log n)$ time, Jarvis presented an alternative solution to the convex hull problem that runs in time $O(nH)$ where H is the number of vertices on the hull [184]. Therefore, if $H = o(\log n)$, Jarvis' algorithm is better than Graham's. The approach taken by Jarvis is suggestive of the idea of "gift-wrapping," also applicable to the technique of [68]. Starting with a point u that is known to be on the convex hull (as in Graham's scan), in linear time we find the next point v such that the edge $\overline{u, v}$ is on the convex hull, i.e., all the remaining points must lie on one side of the directed line containing $\overline{u, v}$. After v has been found, the same technique is applied to locate the next point w such that $\overline{v, w}$ is a hull edge, and so on until we "wrap" back to the starting point u. It is easy to see that the number of iterations needed is exactly the number of vertices on the convex hull, so the total time is $O(nH)$ since each iteration costs $O(n)$ time.

Another interesting method, susceptible of a 3-dimensional generalization, is based on divide-and-conquer [293], [315]. After subdividing the given point set into two subsets of approximately equal size, and recursively finding their convex hulls, the merge step is represented by the following problem.

Problem 3.1.2 (Union of two disjoint convex polygons): Given two disjoint convex polygons P and Q with p and q vertices, respectively, find their convex hull.

This problem is solved in [293] by finding the common supporting lines of P and Q where a *supporting line* of a polygon P is a straight line that has at least one point in common with P and all vertices of P lying on the same side. Since P and Q are, by hypothesis, disjoint, P and Q have just two supporting lines with both P and Q on the same side, which can be found in time $O(p + q)$.

By shrinking Q to a single point, we obtain a straightforward solution of the following additional problem.

Problem 3.1.3 (Supporting lines of a convex polygon): Given a convex polygon P with n vertices and a point u external to P, find the two supporting lines of P passing by u.

This problem, which is trivial without preprocessing, becomes more interesting in the repetitive mode, and can be solved in time $O(\log n)$ by a clever adaptation of binary search [316].

Some of the ideas presented above were elegantly coalesced in a new interesting convex hull technique recently developed by Kirkpatrick and Seidel [198]. Their method is a combination of the divide-and-conquer and prune-and-search approaches, and runs in time $O(n \log H)$, which is also asymptotically optimal [199]. The algorithm constructs separately the upper and the lower hulls, i.e., the two hull chains delimited by leftmost and rightmost vertices. Because of symmetry we shall only consider the upper hull. At first, the point set S is subdivided into two subsets of approximately equal sizes; but then, instead of recursively constructing the upper hull of the two halves and computing their common supporting line (referred to as their *upper bridge*), they first construct this bridge and then separately construct the upper hulls of the subsets, respectively, to the left and to the right of the leftmost and rightmost endpoints of the bridge (*bridge points*). The key of the technique is an efficient way to construct the bridge, to be discussed below; first we give a less informal sketch of the algorithm. If M_1 and M_2 denote the indexes of the points of S with the leftmost and rightmost abscissae (all abscissae are assumed distinct for simplicity), the upper hull is constructed by a call CONNECT($M_1, M_2; S$) of the following procedure:

Procedure CONNECT (min, max; A)
begin
1. Find a vertical line $x = m$ dividing A into two halves.
2. (i, j) := BRIDGE(A, m)
   (**Comment**: i and j are the indexes of the bridge points p_i and p_j lying, respectively, to the left and right of the vertical line.)
3. Let $A_1 = \{p \text{ in } A \text{ such that } x(p) \leq x(p_i)\}$
   Let $A_2 = \{p \text{ in } A \text{ such that } x(p) \geq x(p_j)\}$
4. **If** i = min **then** print (i)
   else CONNECT(min, i; A_1);
   if j = max **then** print (j)
   else CONNECT(j, max; A_2)

end

The running time of CONNECT depends upon that of the function BRIDGE. The latter, however, is easily recognized to be a linear programming problem in two variables and n constraints. Indeed the bridge belongs to a straight line L^* such that each point of S is below L^* and whose intersection with the line $x = m$ is minimum. Thus, letting L: $y = ax + b$, we have

$$\text{minimize} \quad (ma + b)$$
$$\text{subject to} \quad x_j a + b \geq y_j, \qquad j = 1, 2, \cdots, n.$$

By using the prune-and-search techniques of [118] and [257] outlined in Section II-F, this problem can be solved in $\theta(n)$ time. Noting that the median finding of step 1 of CONNECT also uses time $\theta(n)$ and denoting by h the number of upper

hull edges, the running time $T(n, h)$ of CONNECT is governed by the following recurrence relation:

$$T(n,h) = cn \quad \text{if } h = 1,$$
$$= cn + \max_{h_1+h_2=h}(T(n/2,h_1) + T(n/2,h_2)) \quad \text{if } h > 1,\ c \text{ some constant}.$$

It is not difficult to see that $T(n, h)$ is $O(n \log h)$. Thus, in $O(n \log h)$ time we can obtain the upper hull. By a similar scheme the lower convex hull is computed, so that the upper bound $O(n \log H)$ is readily established for the overall running time. See also [254] for a modification of the algorithm in [198] that gives a better expected-case performance.

In dimensions $d \geq 3$, we have the following results. Reference [8] gives an algorithm for constructing the 3-dimensional convex hull without a timing analysis. [185] gives an $O(nF)$ time algorithm where F is the number of faces on the 3-dimensional convex hull, by an approach similar to [184]. Reference [293] gives an optimal time ($O(n \log n)$) divide-and-conquer algorithm for the 2- and 3-dimensional convex hull problems. [68] and [310] give an algorithm for the convex hull problem in dimensions $d \geq 3$; the algorithm in [185] is just the specialization to $d = 3$ of the technique of [68] and the algorithm in [310] is optimal for even dimensions.

Reference [42] gives a linear expected-time algorithm for the planar convex hull problem. See also [97] for a survey on linear expected-time convex hull algorithms.

We conclude this section with a survey of a few additional problems related to the notion of convex hull.

Problem 3.1.4 (Convex hull of a simple polygon): Given a simple polygon with n vertices in the plane, find its convex hull.

The fact that the points are given as a sequence of vertices on a simple polygon enables us to solve this problem in linear time [48], [168], [214], [249], [334].

Problem 3.1.5 (Convex polygon inclusion): Given a convex polygon P with n vertices, with preprocessing allowed, determine if a query point u is in P.

This problem can be solved quite easily by taking an interior point O of P as the origin, arranging the vertices of P in a binary search tree (since the vertices of P are ordered around O), and performing a binary search to locate the sector in which the query point lies. (A sector is the region defined by the origin O and two consecutive vertices of P.) Once the sector is identified, a single comparison against the edge of P in the sector suffices to determine if the point is in the interior of the polygon.

Problem 3.1.6 (On-line convex hull): Given n points, $p_1, p_2, \cdots, p_n$, arriving in sequence one at a time, find the convex hull of $\{p_1, p_2, \cdots, p_i\}$ when p_i arrives.

A straightforward approach which computes the convex hull each time a new point arrives is certainly correct but runs in time $\sum_{i=2}^{n} O(i \log i)$ which is $O(n \log n)$. The question is whether we can do better. It turns out that, by using convexity, an $O(n \log n)$ time algorithm can be designed, with $O(\log n)$ time being sufficient to construct the new convex hull for each newly arrived point [291]. The main idea is to arrange the vertices of the current convex hull in a binary tree of some kind, and for each new point p_i to determine first if it is inside the current hull. If it is, we ignore it; otherwise, we find the two supporting lines from p_i to the current hull. The hull is then updated by removing the appropriate subset of vertices comprised between the two supporting lines and introducing p_i as a new hull vertex. The polygon inclusion test, the identification of the supporting lines, and the removal of nonhull vertices can all be done in $O(\log i)$ time. For details see [291]. A simpler alternative method can be found in [21], [221].

A natural question to address at this point is what happens when deletions of points are allowed. In this case we can no longer obliterate those points that have been declared not on the convex hull since they may reappear as hull vertices when a hull point is deleted. Thus, we have the following problem.

Problem 3.1.7 (Maintenance of convex hull): Given a sequence of points, $p_1, p_2, \cdots, p_n$, some of which correspond to deletions and some of which correspond to insertions, maintain their convex hull.

This problem has been elegantly solved by Overmars and van Leeuwen [281]. They have shown that the convex hull can be maintained in $O(\log^2 n)$ time per insertion or deletion.

Problem 3.1.8 (Depth and convex layers of a set): Given a set S of n points in the plane, find the sequence $(CH(S_i) \mid i = 1, 2, \cdots, d)$ where $S_0 = S$, and $S_i = S_{i-1} - V(CH(S_{i-1}))$, $i = 1, 2, \cdots, d$ where $V(CH(T))$ denotes the set of vertices of $CH(T)$, and $S_d = V(CH(S_d))$. The depth of the set is d, i.e., the number of convex layers resulting from the above process.

The problem arises in statistical estimation, where the "observations" (points) lying on the outer layers of convex hulls represent outliers or "noise" and should be excluded from being used for estimation purposes. An $O(n^2)$ algorithm can be easily obtained by modifying Jarvis' algorithm. With Overmars and van Leeuwen's algorithm [281] the worst case bound can be improved to $O(n \log^2 n)$. Recently, Chazelle [75] has obtained an optimal $O(n \log n)$ algorithm for this problem. The result has an application in half-planar range searching problem and in computing the *depth* of any given point. (The *depth* in S of a query point is defined to be the number of convex layers of S that enclose it.) By preprocessing the convex layers, the depth of a query point can be determined by locating the point in the regions between consecutive layers (cf. Problem 3.3.10).

No efficient algorithm, other than the obvious one, is known for computing the convex layers of a point set in dimensions $d \geq 3$.

B. Intersections

Intersection problems and their variations arise in many disciplines, such as architectural design, computer graphics, pattern recognition, etc. An architectural design cannot place two inpenetrable objects to share a common region. When displaying objects on a 2-dimensional display device, obscured portions (or intersecting portions) should be eliminated to enhance realism, a long standing problem known as hidden line/surface elimination problem [268]. In integrated

circuit design two distinct components must be separated by a certain distance, and the detection of whether or not the separation rule is obeyed can be cast as an instance of intersection problems; since the task may involve thousands of objects, fast algorithms for detecting or reporting intersecting or overlapping objects are needed. Another motivation for studying the complexity of intersection algorithms is that light may be shed on the inherent complexity of fundamental geometric problems. For example, how difficult is it to decide if a given polygon with n vertices is simple (**simplicity test**) or how much time is needed to determine if any two of n given objects in the plane, such as polygons, line segments, etc., intersect (**intersection detection**)? Both problems are solvable in $O(n \log n)$ time [317]; this result is optimal under the algebraic computation tree [26] model (with certain primitive operations) for the latter problem, whereas for the former problem it is still open whether $\Omega(n \log n)$ is a lower bound.

standard form, i.e., $P = (v_0, v_1, \cdots, v_{n-1})$ and (v_i, v_{i+1}) is an edge, for $i = 0, 1, \cdots, n-1, v_n = v_0$, so that the interior of the polygon lies to the left as the edges are traversed. answer as to which two objects intersect if they intersect at all). It is plausible that detection is easier than construction, and substantiating evidence is provided in [82]. We now discuss a category of significant intersection problems. In the following we assume that when a simple polygon P is given, the polygon is specified by a sequence of vertices in its *standard* form, i.e., $P = (v_0, v_1, \cdots, v_{n-1})$ and (v_i, v_{i+1}) is an edge, for $i = 0, 1, \cdots, n-1, v_n = v_0$, so that the interior of the polygon lies to the left as the edges are traversed.

Problem 3.2.1 (Intersection of convex polygons): Given two convex polygons P and Q with m and n vertices, respectively, compute their intersection.

This problem can be solved by the plane-sweep technique in $O(m + n)$ optimal time [317] by observing that the intersection of each polygon with the vertical strip comprised between two successive event points is, in general, a trapezoid (possibly empty or degenerating to a triangle), and that the computation of the intersection of two trapezoids takes $O(1)$ time (Fig. 3). An alternative method has been proposed in [273].

Problem 3.2.2 (Intersection of star-shaped polygons): Given two star-shaped polygons with m and n vertices, respectively, find their intersection. (A polygon P is a star-shaped if there exists at least one point q in P such that every point of P is *visible* from q, i.e., the line segment connecting q and any point of P lies entirely in P.)

This problem, however, requires $\Omega(mn)$ time since the intersection may consist of $O(mn)$ disjoint components. Thus, a general polygon intersection problem requires quadratic time in the worst case.

Problem 3.2.3 (Segment intersection detection): Given n line segments in the plane, determine if any two intersect.

This problem can be solved by the plane-sweep technique in $O(n \log n)$ time [317]. The algorithm is based on the following idea. Consider a vertical sweep line L. L cuts some of the line segments, and these intersections form a totally

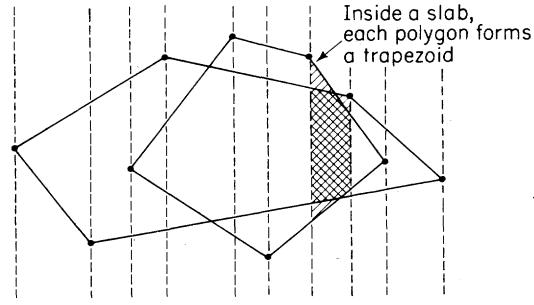

Fig. 3. Vertical strips defined by the vertices of two convex polygons.

ordered set. This order is maintained in any vertical strip not containing intersections or segment endpoints. If any two segments intersect, then they must be adjacent to each other during the plane-sweep process. Thus, whenever a line segment becomes "active," the line segment is checked against its two neighbors for intersection; and whenever a line segment terminates, its two neighbors (they are now adjacent) are checked for intersection. Since we perform at most n insertions and n deletions, and each operation can be solved in $O(\log n)$ time (the intersection check takes $O(1)$ time), the total time is $O(n \log n)$, which is optimal (cf. Lemma 3.6.4). It follows that the **simplicity test problem**, i.e., determining if a given polygon is simple, takes $O(n \log n)$ time, and the problem of determining if any two of n given circles intersect can also be solved in $O(n \log n)$ time.

Problem 3.2.4 (Polygon intersection detection): Given two simple polygons P and Q with m and n vertices, respectively, determine if they intersect.

Two polygons intersect if either an edge of one polygon intersects an edge of the other or one is totally contained inside the other. The former can be detected in $O(N \log N)$ time where $N = m + n$, and the latter can be solved in $O(m + n)$ time, by taking an arbitrary vertex of P and Q and checking for inclusion in the other polygon since if polygon P contains polygon Q, it must contain any vertex of Q. Thus, the total time is $O(N \log N)$.

(The **point enclosure problem**, i.e., determining if a point p lies in a polygon Q, can be viewed as an intersection problem, i.e., is $p \cap Q = \emptyset$, and can be solved in linear time [313].) Note that if both P and Q are convex, the problem can be solved in $O(\log N)$ time [82].

Problem 3.2.5 (Segment intersection reporting): Given n line segments find all intersecting pairs.

This is a typical intersection reporting problem. The plane-sweep technique works here equally well. An $O((n + k) \log n)$ time and $O(n + k)$ space algorithm where k is the number of intersecting pairs reported, can be found in [38]. By carefully implementing the event point schedule which is changing over time, Brown [63] has shown that the storage space can be reduced to $O(n)$. Recently, Chazelle[76] has obtained an $O(n \log^2 n / \log \log n + k)$ time and $O(n + k)$ space algorithm, the first known solution of the general problem with time complexity proportional to the output size k.

Problem 3.2.6 (Segment intersection counting): Given n line segments find the *number* of intersecting pairs.

This problem is different from Problem 3.2.5 in that it only calls for counting rather than reporting intersecting pairs. Obviously, any construction algorithm can be used to solve the corresponding counting problem. But since the parameter k, which is the output size, can be too large ($O(n^2)$ in our case), it may not be efficient. Chazelle in the same paper [76] cited earlier has obtained an $O(n^{1.695})$ time and $O(n)$ space algorithm.

Problem 3.2.7 (Half-plane intersection construction): Given n half-planes in the plane, compute their common intersection.

One can show that the **intersection of n half-planes** can be found in $O(n \log n)$ time by divide-and-conquer since the conquer step is just an instance of Problem 3.2.1, which can be solved in linear time. The bound is optimal (see Section III-F).

Problem 3.2.8 (Kernel of a polygon): Given a simple polygon with n vertices, find its kernel. (The *kernel* is the intersection of n half-planes, each of which lies to the left of the line containing a polygon edge and oriented as the edge in the standard representation of the polygon.)

Certainly, the kernel can be found in $O(n \log n)$ time by simply applying the algorithm for computing the intersection of half-planes. However, the fact that these n half-planes are not arbitrary but "determined" by the edges of the simple polygon enables one to obtain a $\theta(n)$ algorithm [227].

Problem 3.2.9 (Convex polyhedra intersection construction): Given two convex polyhedra P and Q with m and n vertices, find their intersection.

This is yet another example for which the geometric transformation technique, known as duality, can be successfully resorted to. Muller and Preparata [265] have obtained an $O(N \log N)$ time algorithm where $N = m + n$. Dyer also has developed an $O(N \log N)$ algorithm for this problem [117]. An entirely different and elegant space-sweep technique has been recently proposed in [176] to solve this problem within the same time bound.

Problem 3.2.10 (Convex polyhedra intersection detection): Given two convex polyhedra P and Q with m and n vertices, respectively, determine if they intersect.

The algorithms for the corresponding intersection construction problem certainly work for this case. However, since we only need to provide a witness (a point in the intersection) to this problem if indeed the polyhedra intersect, we expect to have a faster solution. Dobkin and Kirkpatrick give an $O(n)$ algorithm for this problem in [104], in which they also credit Dyer with an independently discovered linear time solution. If the polyhedra have been preprocessed, sublinear (in fact, polylogarithmic) solutions can be obtained [82], [103], [105]. Similar statements can be made about the convex polygon intersection detection problem. See [2], [57] for other methods for interference detection among solids.

Problem 3.2.11 (Half-space intersection construction): Given n half-spaces in three dimensions, compute their common intersection.

This can obviously be solved by divide-and-conquer in $O(n \log^2 n)$ time since the conquer step is an instance of Problem 3.2.9, which can be solved in $O(n \log n)$ time. Preparata and Muller [294] make use of the geometric transformation technique and the separating half-plane algorithm (see Problem 3.5.2) to obtain an optimal $O(n \log n)$ algorithm for this problem.

Problem 3.2.12 (Rectangle intersection reporting): Given n isothetic rectangles, i.e., rectangles whose sides are parallel to coordinate axes, find all k intersecting pairs.

This problem arises in the design of VLSI circuitry. Each rectangle is used to model a circuitry component and certain design rules must be adhered to [24], [35], [46], [126], [207], [250], [251], [320]. Optimal ($O(n \log n + k)$ time and $O(n)$ space) algorithms have been obtained independently by McCreight [250] and Edelsbrunner [123]. They both use the plane-sweep technique in connection with a novel data structure, called *interval tree,* to maintain the sweep-line status. The same time and space bounds can be achieved using the divide-and-conquer technique [174], [220]. Making use of *segment trees* [28], [292], Chazelle and Incerpi [87] also develop an algorithm with the same bounds. Furthermore, the technique, called *unravelling* of segment trees in [87], can be generalized to higher dimensions as discussed below. By transforming this problem into a batched sequence of queries, insertions, and deletions Edelsbrunner and Overmars [136] also obtain similar results.

Problem 3.2.13 (d-Range intersection reporting): Given n isothetic hyperrectangles in d-dimensional space, $d > 2$, find all k intersecting pairs.

Six and Wood obtained an $O(n \log^{d-1} n + k)$ time and $O(n \log^{d-1} n)$ space algorithm [321], whose time bound was later improved by Edelsbrunner [126] to $O(n \log^{d-2} n)$. Within the same time bound Chazelle and Incerpi [87] and Edelsbrunner and Overmars [136] have reduced the space bound down to $O(n)$, which is optimal.

Problem 3.2.14 (d-Range intersection counting): Given n isothetic hyperrectangles in d-dimensional space, $d \geq 1$, find the *number* of intersecting pairs.

Unlike the segment intersection counting problem, this problem can be solved in $O(n \log^{d-1} n)$ time and $O(n \log^{d-2} n)$ space by slightly modifying the intersection reporting algorithm [321]. We remark that the algorithm in [87] cannot be adapted to solve the counting problem efficiently.

Problem 3.2.15 (Rectangle containment reporting): Given n isothetic rectangles, find all k containment pairs.

This problem is first studied in [336] where an $O(n \log^2 n + k)$ time and $O(n \log^2 n)$ space algorithm is reported. Lee and Wong [234], and independently Edelsbrunner [122], obtained the same result using a geometric transformation that maps a d-dimensional problem to a $2d$-dimensional problem, for $d \geq 1$. Lee and Preparata [228] later used divide-and-conquer to improve the space bound to $O(n)$. This problem has been shown to be equivalent to the 4-dimensional dominance problem [135], [228] (cf. Section II-C).

Problem 3.2.16 (Polygon containment detection): Given two polygons P and Q with m and n vertices, respectively, with rotation or translation allowed determine if Q contains P.

Chazelle[71] has obtained several algorithms for this problem, involving general polygons, convex polygons, etc. If both P and Q are general polygons, it can be solved in $O(m^3n^3(m + n) \log (m + n))$ time; and when Q is convex, the problem can be solved in $O(mn^2)$ time.

Problem 3.2.17 (Visibility polygon from a point): Given a set of disjoint polygons in the plane and a point q, find the portion of the boundary of these polygons that is visible from q.

This is an instance of the hidden-line elimination problem in computer graphics. The hidden-line or hidden-surface problem has been extensively studied [65], [152], [153], [159], [326]. If the set consists of a single simple polygon, at least two linear time algorithms are known [143], [218]. The technique used is similar to Graham's scan in that an initial point visible from q is first determined and then by using the property of simplicity, the vertices of P are scanned and the portions of P that are not visible from q are eliminated. In the case where the set consists of m disjoint convex polygons with n edges in total and F denotes the output size, this problem can be solved in $O(m \log n + F)$ [221]. Edelsbrunner *et al.* [138] also examine this problem and discuss the maintenance of the visibility polygons when insertions or deletions are allowed, as well as the issue of coherence when the viewing direction or the points are allowed to move.

Problem 3.2.18 (Visibility polygon from an edge): Given a simple polygon P with n vertices and an edge e of P, find the portion of the boundary of P that is visible from e. (A point q in P is *visible from the edge* e if there exists a point r on e such that the line segment $\overline{q, r}$ does not intersect the boundary of P.)

The edge-visibility problem was first studied by Avis and Toussaint [22]. They presented an $O(n)$ algorithm for determining if P is visible, called *weakly visible* in [22], from an edge e. It is shown in [22] that the boundary of P is visible from e if and only if the interior of P is visible from e, while the problem of determining the visibility polygon in $O(n)$ time is posed as an open problem. An $O(n \log n)$ algorithm has been obtained independently by El Gindy [142] and Lee and Lin [223]. Chazelle and Guibas [85] have also obtained an $O(n \log n)$ time algorithm using the polygon cutting theorem [70].

There are other problems pertaining to the measure, the perimeter, the contour, or the number of connected components of a union of isothetic rectangles or other geometric objects, such as squares and circles [87], [134], [172], [178], [179], [182], [183], [239]–[241], [323], [342]. However, the intersections between sets of objects have not received adequate attention. For example, given two sets of line segments, find all intersecting pairs whose members do not belong to the same set. A recent article by Mairson and Stolfi [246] reports an optimal algorithm, i.e., $O(n \log n + k)$ time and $O(n)$ space, for this problem where it has been assumed that no two segments of the same set intersect. Thus, it follows [246] that the intersection of two simple polygons can be found in $O(n \log n + k)$ time.

C. Geometric Searching Problems

In this section we shall present a collection of geometric searching problems. A *searching* problem consists of *querying* a certain database and *gathering* a *response* from the database. The database is composed of certain geometric objects and is normally structured so as to aid the searching process. There are two types of queries, one-time queries and repetitive queries, as classified by Preparata and Shamos [295]. The former type is referred to as *single shot,* and the latter as *repetitive mode.* Depending on whether the database is fixed or susceptible of being updated, we have two types of environments, *static* and *dynamic,* respectively. Single-shot queries are of less interest than the repetitive mode ones. In the latter, we have basically four cost measures in terms of the size n of the database, and of the size F of the query response.

1) Query time $Q(n, F)$, the time needed to resond to a query.

2) Storage $S(n)$, the amount of space required by the structured database.

3) Preprocessing time $P(n)$, the time needed to organize the database into suitable forms to facilitate searching.

4) Update times, insertion time $I(n)$ and deletion time $D(n)$, the times needed to insert or delete an entry, respectively. These measures are meaningful only for dynamic databases; if $I(n) = \theta(D(n))$, then we refer to the update time $U(n) = O(I(n)) = O(D(n))$.

The problems to be described next are formulated in the plane setting. Generalizations to the d-dimensional space will be mentioned where appropriate; in this case the cost measures will bear a subscript d each.

Problem 3.3.1 (Range search counting): Given n points in the plane, find the number of points that lie in a given (query) rectangle, specified as a range product $(a_1, a_2) \times (b_1, b_2)$. That is, find the number v of points $p = (x, y)$ such that $a_1 \leq x \leq a_2$, $b_1 \leq y \leq b_2$. (Note that F, the response size, is a single integer.)

This problem can be solved by the locus method discussed in Section II-B. Let the corners of the query rectangle be A, B, C, and D, with coordinates $(a_1, b_2), (a_1, b_1), (a_2, b_1)$, and (a_2, b_2), respectively, and denote by $\mathrm{Dom}(q)$ the number of points dominated by a point q. Since the number of points in the query rectangle is equal to $\mathrm{Dom}(D) - \mathrm{Dom}(A) - \mathrm{Dom}(C) + \mathrm{Dom}(B)$, four searches locating the corners of the query rectangle in the $O(n^2)$ *cells* of the plane are sufficient. (Locating a point consists of *two* binary searches, one locating the point in the vertical strips and the other in the horizontal strips.) Thus, we have $Q(n) = O(\log n)$, $S(n) = P(n) = O(n^2)$. In d dimensions the locus method, or cell method, generalizes to yield the result: $Q_d(n) = O(\log n)$ and $S_d(n) = P_d(n) = O(n^d)$.

Bentley introduced a data structure, called *range tree,* to solve range searching problems. With this structure Lueker and Willard [244], [344], [347], Lee and Wong [233], and Bentley and Shamos [41] were able to obtain the following result for the range search counting problem in d dimensions, $d \geq 2$: $Q_d(n) = O(\log^d n)$, $P_d(n) = S_d(n) =$

$O(n \log^{d-1} n)$. The range tree structure will be described below.

Problem 3.3.2 (Range search reporting): Given n points in the plane, *report* all points that lie in a given (query) rectangle. (The query is specified in the same way as in Problem 3.3.1.)

This problem has an obvious application in database systems. In a database of the records of the employees of a company, each record may have multiple attributes, such as age, salary, etc., and can be considered as a point in the d-dimensional space where each attribute is identified with a coordinate and d is the number of attributes. A typical range search problem in two dimensions consists of retrieving all the employees whose ages are within a given range and whose salaries are within another. There are a number of research contributions to this subject. A survey of algorithms and data structures for range searching by Bentley and Friedman describes the state-of-the-art as of 1979 [34]. A result for this problem and also for its counting counterpart, which is based on the range tree, is due to Willard [346] and has the following performance: $Q_d(n, F) = O(\log^{d-1} n + F)$, $P_d(n) = S_d(n) = O(n \log^{d-1} n)$. See [158] for a different technique achieving the same bounds. The space bound has later been improved by a log log n factor by Chazelle [74]. This improvement, however, holds only for the reporting problem.

The range tree structure suggests the shape of a "pyramid" and is commonly referred to as a tree of trees. We shall start with the structure for the 1-dimensional case and then define recursively the trees for the d-dimensional case from the trees for the $(d-1)$-dimensional case. The 1-dimensional range tree is a sorted array, i.e., an array or list of records sorted in increasing order of the values of the first coordinate. The range tree in two dimensions is an ordinary balanced binary search tree arranged according to the *second coordinate* of each record. Each leaf is assigned one record and each node is assigned the records in its subtree; in addition, each node is augmented with a 1-dimensional range tree, organizing the records assigned to the node according to their *first coordinate*. This shows that at level i the n records are approximately equally partitioned among 2^i nodes; this partitioning continues for approximately $\log_2 n$ levels, and is conveniently stopped when the record sets are so small that they can be comfortably searched by brute force. The same idea generalizes to higher dimensions, so that in d dimensions we have a balanced binary search tree on the dth coordinate and each node is associated with a $(d-1)$-dimensional range tree on the $(d-1)$-dimensional records assigned to the node.

The search algorithm is recursive, and, without loss of generality, we shall describe the 2-dimensional case. Each node in the tree represents a range in the y-dimension. When visiting a node we first compare the y-range of the query to the y-range of the node. If the node's y-range is entirely within the query's, then we search the sorted array stored at the node for all points in the query's x-range. If the query's y-range is entirely below the discriminating value of the node, we recursively visit the left subtree. If the query's range is entirely above the discriminating value, we recursively visit the right subtree. Otherwise (the range contains the discriminating value), we visit both subtrees.

The analysis of the planar range tree is somewhat complicated. Since there are $\log_2 n$ levels and each level has n points, the total storage required is $O(n \log n)$. (Preprocessing can also be performed in $O(n \log n)$ time.) As for the query time, the analysis shows that at most two sorted array searches are done at each level of the tree, each of cost $O(\log n)$, so the total cost is $O(\log^2 n)$, plus the retrieval time. Thus, we have $P(n) = S(n) = O(n \log n)$ and $Q(n, F) = O(\log^2 n + F)$. Generalizing the above arguments, we have for the d-dimensional range search reporting problem the following: $P_d(n) = S_d(n) = O(n \log^{d-1} n)$ and $Q_d(n, F) = O(\log^d n + F)$.

A clever observation leads to a log n factor improvement in the query time. Referring again to the 2-dimensional case, it is readily realized that there is considerable redundancy in the search of the sorted arrays for the x-dimension since the set of each node is a subset of the sets of its ancestors. With this idea, the arrays of all nodes, save the root, are replaced by lists; the x-range search is done exclusively in the 1-dimensional structure associated with the root, and location in each offspring is done by pointers. In this manner $Q_d(n, F) = O(\log^{d-1} n + F)$ can be achieved [344], [346], [347].

Other data structures for the range search reporting problem can be found in [27], [37], [43], [231].

We note that data structures discussed above do not allow insertions or deletions of records. However, by using the dynamization techniques (cf. Section II-G) the range tree structure can be adapted to work in a dynamic environment since the range searching problems are *decomposable* in the sense of Bentley [29]. If only insertions are allowed starting from an empty database, it is shown in [39], [304] that the following performance can be achieved: $P_d(n) = O(n \log^d n)$, $S_d(n) = O(n \log^{d-1} n)$, and $Q_d(n, F) = O(\log^{d+1} n + F)$ where n is the current number of executed insertions. In [244] Lueker and Willard have improved the query time by a log n factor while keeping the other two measures unaltered even when deletions are allowed. In [347] it is shown how to guarantee fast times for each individual update, rather than only over a sequence of such operations. Specifically, the following performance can be obtained: $Q_d(n, F) = O(\log^d n + F)$, $S_d(n) = O(n \log^{d-1} n)$, and insertion and deletion times are $I_d(n) = O(\log^d n)$, and $D_d(n) = I_d(n)/\log n$, respectively [283]. By increasing the deletion time by a log n factor Edelsbrunner was able to reduce the query time by a log n factor [125]. There are other dynamization results with various query-time/space or query-time/update-time tradeoffs. (Consult the list of references given in Section II-G.) Fredman has shown that, under a restricted model, $\Omega(n \log^d n)$ is a lower bound on the complexity of performing sequences of n updates and retrievals [151]. Instead of using tree structures to mechanize the range searching problem, Bolour has described a hashing technique, called box-array hashing in [52], and obtained reasonably fast solutions, on the average,

to the range searching problem when the query domain is within prescribed bounds.

Problem 3.3.3 (Polygon-range retrieval): Given n points in the plane, with preprocessing allowed, find all the points within a given query polygon with k sides.

This is a fairly general searching problem. In [345] Willard has devised a data structure, called *polygon tree,* and has shown that the problem can be solved in time $Q(n,F) = O(kn^{0.77} + F)$ and $P(n) = O(n^2)$ in the worst case (this can be reduced to $P(n) = O(n \log^2 n)$ in the expected case); the storage space is $S(n) = O(n)$. In the case of a counting search, $O(kn^{0.77})$ query time also suffices. These results have been improved to $Q(n,F) = O(kn^{0.695} + F)$ and $O(n^{0.695})$ for the reporting and counting problems, respectively, using a new data structure, called the conjugation tree [141].

An interesting simplification of the problem is the **half-plane retrieval problem** [140], in which the query polygon becomes a half-plane. Previous results for this problem include [131] with $S(n) = O(n^3)$, $Q(n,F) = O(\log n + F)$, [141] with $S(n) = O(n)$, $Q(n) = O(n^{0.695} + F)$, and [345], of course. A most recent result [86] that combines the idea of geometric transforms and convex layers (Problem 3.1.8) shows that the problem can be solved in optimal time $Q(n) = O(\log n + F)$ with $S(n) = O(n)$ and $P(n) = O(n \log n)$. However, the technique of [86] cannot be used to solve the corresponding counting problem within the same bounds. This question remains open. For half-space retrieval problem Yao [353] has obtained a solution wih $S(n) = O(n)$ and $Q(n,F) = O(n^{0.93} + F)$ using octant trees; the query time has later been improved to $O(n^{0.916})$ by Dobkin and Edelsbrunner [101]. With $O(n^4)$ space the problem can be solved in $O(\log n + F)$ time [95]. In fact, Cole and Yap [95] have shown that with $O(dn^{2d-1})$ space, $O(2^d \log n)$ time suffices.

Problem 3.3.4 (Fixed-disk retrieval): Given n points in the plane, with preprocessing allowed, find all the points within a query disk of fixed radius and arbitrary center.

This problem is also known as the fixed-radius near neighbor searching problem. Using the locus method, Bentley and Maurer [36] gave an algorithm with the following performance: $Q(n,F) = O(\log n + F)$, $P(n) = S(n) = O(n^3)$. Chazelle [73] has improved both the preprocessing time and storage to $O(n^2 \log n)$. The space requirement is further improved to $O(n^2)$ [128]. Most recently, with $O(n^2)$ preprocessing time Chazelle and Edelesbrunner [84] have reported an optimal ($O(n)$ space and $O(\log n + F)$ time) algorithm for this problem.

Problem 3.3.5 (Variable-disk retrieval): Given n points in the plane, with preprocessing allowed, find all the points within a disk of arbitrary radius and center.

Yao has obtained an $O(n^{0.98} + F)$ query time algorithm with linear space and $O(n^4)$ preprocessing time [353], the first technique exhibiting an $o(n)$ worst case time. Cole and Yap [95] have obtained an $O(n \log^3 n)$ space and $O(\log n + F)$ time algorithm; the space bound has been improved to $O(n(\log n \log\log n)^2)$ [80].

In the above range searching problems the objects in the database are points of space. Interesting range searching problems arise when the geometric objects are of more complex nature, such as line segments or polygons. Below we shall consider these significant generalizations.

Problem 3.3.6 (Range search reporting in a set of line segments): Given a set of n line segments in the plane, find all the line segments that have nonempty intersection with a given rectangle.

This problem is known as window clipping in computer graphics [268]. Overmars [277] and Edelsbrunner *et al.* [137] have obtained an elegant solution for the problem that even allows updates with $Q(n,F) = O(\log^2 n + F)$, $U(n) = O(\log^2 n)$, and $S(n) = O(n \log n)$. If the window is to move in a fixed direction parallel to one of the coordinate axes, the line segments whose status with respect to the window has changed can be determined in $Q(n,k) = O(\log^2 n + k)$ where k is the total number of such segments.

Problem 3.3.7 (Orthogonal intersection searching): Given a set of n orthogonal objects, find all the objects that intersect a given orthogonal object. (An *orthogonal* object of dimension d is the Cartesian product of d intervals, each of which may reduce to a single value. For instance, a point, an isothetic rectangle, and a horizontal or vertical line segment are orthogonal objects.)

This class of searching includes Problems 3.3.1 and 3.3.2. The *inverse range searching,* or *point enclosure* problem, i.e., given n isothetic rectangles, find the rectangles that enclose a query point, also belongs to the class. Vaishnavi [335] gives a data structure that supports searching in $O(\log n + F)$ query time with $P(n) = S(n) = O(n \log^2 n)$. Chazelle [74] gives an optimal algorithm for this problem, i.e., $O(n)$ space and $O(\log n + F)$ time. Using the dynamization technique of Overmars and van Leeuwen [279], [282], the dynamic point enclosure problem can be solved in $Q(n,F) = O(\log^3 n + F)$ time with $I(n) = O(\log^3 n)$, $D(n) = O(\log^2 n)$, and $S(n) = O(n \log^2 n)$. The *orthogonal line segment intersection searching* problem requests, for given n horizontal and vertical line segments, to find all segments intersecting a given orthogonal segment, and was investigated by Vaishnavi and Wood [337], who gave an $O(\log n + F)$ query time algorithm with $O(n \log n)$ preprocessing and space. The space bound can be reduced to $O(n)$ [74]. If the set of distinct coordinate values is of cardinality $O(n)$ and when insertions or deletions of segments are allowed, McCreight presents a dynamic data structure [251] for this problem which requires $O(n)$ space with $Q(n,F) = O(\log^2 n + F)$ and $U(n) = O(\log^2 n)$; and Lipski and Papadimitriou [239] provide an algorithm with $Q(n,F) = O(\log n \log\log n + F)$, $U(n) = O(\log n \log\log n)$, and $S(n) = O(n \log n)$. For the general dynamic problem Edelsbrunner [124] gives an algorithm with $Q(n,F) = O(\log^2 n + F)$, $U(n) = O(\log^2 n)$, and $S(n) = O(n \log n)$. The *rectangle intersection searching* problem, however, deals with isothetic rectangles, and is considered by Lee and Wong [234] and Edelsbrunner [122, 123]. By transforming this problem in d dimensions to an instance of a range search reporting problem in $2d$ dimensions, Lee and Wong [234] give an algorithm with $Q_d(n,F) = O(\log^{2d-1} n + F)$, $P_d(n) =$

$S_d(n) = O(n \log^{2d-1} n)$. Edelsbrunner [122], [123] generalizes to d dimensions his results on the reporting of all intersecting pairs of rectangles (cf. Problem 3.2.12) and improves the preprocessing and space bounds of [234] to $P_d(n) = O(n \log^d n)$ and $S_d(n) = O(n \log^{d-1} n)$.

Edelsbrunner and Maurer [129] have recently unified all the previous approaches to solving this class of intersection searching problems and obtained the following results. For the static problem, there exists a data structure achieving the following performance: $Q_d(n, F) = O(\log^{d-1} n + F)$ and $P_d(n) = S_d(n) = O(n \log^d n)$. The space bound has been improved to $O(n \log^{d-1} n / \log \log n)$ [85]. For the dynamic problem, there exists a dynamic data structure achieving the following performance: $Q_d(n, F) = O(\log^d n + F)$, $S_d(n) = O(n \log^d n)$, and $U_d(n) = O(\log^d n)$. See also [124], [158] for more details.

Problem 3.3.8 (Stabbing set and stabbing number): Given a set of n objects in the space and a query object, report the set of objects intersecting the query object (stabbing set) or simply find its cardinality (stabbing number) [173].

The choice of such exotic name comes from its 2-dimensional instance where the objects (e.g., polygons) lie in the plane and the query object is a point identified with a "needle" orthogonal to the plane. If the objects are orthogonal, then we rediscover the orthogonal intersection reporting and counting problems, respectively. In [131] general solutions with $Q(n, F) = O(\log n + F)$ and $Q(n) = O(\log n)$ for the stabbing set and stabbing number problems, respectively, are given, but preprocessing costs and times are very high. In 1-dimensional space if the objects are intervals and query is a point, optimal solutions, i.e., $O(\log n + F)$ query time for the stabbing set problem [74], [122], [250] and $O(\log n)$ for the stabbing number problem [129], [164] have been obtained. Several instances of this problem are currently being investigated. For example, the stabbing number problem for a set of n polygons whose sides have a constant number of orientations can be solved in $O(\log n)$ query time and $O(n \log n)$ preprocessing and storage [173]. Edelsbrunner *et al.* [132] have given an algorithm for finding a stabbing line which intersects each of n given line segments. The algorithm runs in $O(n \log n)$ time and requires $O(n)$ space; an $O(\log n)$ time algorithm is also given to determine if any given line is a stabbing line. The problems of finding which or how many line segments intersect a given line segment and of finding which or how many polygons with bounded number of edges contain a given query point can be answered in $O(n^{0.695})$ time [102]. Reference [74] gives an algorithm for reporting the line segments intersecting a given line segment with $P(n) = O(n^2 \log n)$, $S(n) = O(n^2)$, and $Q(n, F) = O(\log n + F)$.

Problem 3.3.9 (Polygon intersecton searching): Given a set of simple polygons, each with a bounded number of edges, find the polygons whose intersection with a query polygon of the same type is nonempty.

This is the most general intersection searching problem of its kind, of which most of the problems in the plane given before are special cases. The problem, by suitable transformations [131], can be reduced to three subproblems, each of which is a generalization of its counterpart in orthogonal intersection searching (Problem 3.3.7), i.e., of the inverse range searching problem, of the orthogonal line segment intersection searching problem, and of the range searching problem. The three subproblems are as follows: i) given a point, find its stabbing set of polygons; ii) given a line segment, find its stabbing set of line segments; and iii) given a polygon, find all the points it contains. Problem i) can be solved by point location (cf. Problem 3.3.10 below), problem ii) by geometric transformation, and problem iii) by answering a bounded number of *triangular range search* queries, all in time $O(\log n + F)$. The preprocessing and storage costs can be as high as $O(n^7)$ [131]. Recently Dobkin and Edelsbrunner [102] have studied this problem and obtained sublinear query-time solutions.

Problem 3.3.10 (Point location): Given a planar subdivision with n edges, find the region of the subdivision that contains a query point. (The subdivision can be regarded as a collection of polygons and we look for the polygon whose intersection with the query point is nonempty.)

This is regarded as one of the fundamental searching problems in computational geometry. It arises in many disciplines. In pattern classification, for instance, one wishes to classify a query pattern in a finite set of classes. Patterns are mapped to points of some space, which is partitioned into regions, each corresponding to a specific class. The classification is done by locating the region in which the query pattern lies and the name of the corresponding class will be the answer.

Dobkin and Lipton [106] were probably the first to give an $O(\log n)$ time algorithm for locating a point in a multidimensional subdivision [297]. Their method is based on the idea of partitioning the space into slabs (*slab method*), and is best described in two dimensions (see also [295], [315]). See [78] for a generalization of the Dobkin–Lipton technique [106]. Suppose we have a planar subdivision where each region is delimited by straight-line edges. The graph is referred to as *planar straight-line graph,* PSLG for short, in [295]. Assume for simplicity that no two vertices of the graph have identical ordinate and draw a horizontal line through each vertex. Two such consecutive lines form a *slab* (a horizontal strip of the plane), in which no two edges in the PSLG interesect (Fig. 4). Note that in each slab the edges form a total ordering. The point location problem can be solved by two consecutive binary searches, one locating the query point in a slab and the other locating it within the slab, in $O(\log n)$ total time. Each slab is organized as a binary search tree, with total storage $O(n^2)$; preprocessing can also be done in $O(n^2)$ time [295]. The generalization of the method to higher dimensional space is straightforward. Because of its high preprocessing and storage cost, however, several improvements were later proposed. The chain method of Lee and Preparata [225] runs in $Q(n) = O(\log^2 n)$ time with $P(n) = O(n \log n)$ and $S(n) = O(n)$. Lipton and Tarjan [242] proposed an extremely sophisticated algorithm, based on the planar separator theorem, achieving optimal performance, i.e., $Q(n) = O(\log n)$, $P(n) = O(n \log n)$, and $S(n) = O(n)$. Unfortunately, the Lipton–Tarjan tech-

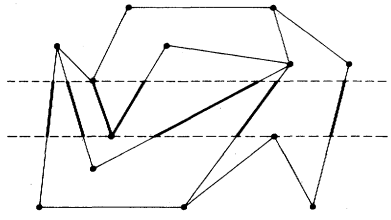

Fig. 4. Within a slab no two segments intersect.

nique is merely of theoretical interest. Preparata [292] later proposed a new scheme, now known as *median-guided trapezoid method,* which solves the problem in optimal query time, i.e., $O(\log n)$, but requires $P(n) = S(n) \doteq O(n \log n)$. Kirkpatrick [197] made use of triangulations to obtain a hierarchical representation of the PSLG with $O(\log n)$ levels of triangulations, each being a refinement of the other, yielding a more practical optimal algorithm with $O(\log n)$ query time, $O(n \log n)$ preprocessing time, and $O(n)$ storage. All of the above techniques, except Kirkpatrick's, can handle planar subdivisions with suitable curvilinear edges; another algorithm with similar capabilities has been proposed by Edelsbrunner and Maurer [130] and has the following performance: $Q(n) = O(\log^3 n)$, $P(n) = O(n \log n)$, and $S(n) = O(n)$. A recent report by Edahiro *et al.* [119] contains a comprehensive comparison, including actual running times, of the above point-location algorithms. Recently, a new optimal technique, and perhaps the most practical method available today, has been proposed in [128]: its key idea is the adaptation of philosophy of layering [346], improved to $O(n)$ space in [74], to the Lee–Preparata chain method [225]. Chazelle [77] has obtained an $O(\log^2 n)$ time and $O(n)$ space algorithm for locating a point in a 3-dimensional complex with $O(n)$ vertices.

D. Proximity and Related Problems

Geometric objects, such as points and circles, are used to model physical entities in the real world. In some cases we would like to have access to a suitable neighborhood of the objects. For instance, in air traffic control we wish to keep track of the closest two aircrafts; when aircrafts are modeled as points moving in space, we want to find the closest pair of points at a certain point in time. We shall first list a number of problems, some of which may appear unrelated, and describe a geometric construct, called a *Voronoi diagram,* which can be used to solve these problems within the same order of the time spent for computing the diagram. Some proximity related problems are given afterwards.

1) Basic Proximity Problems:

Problem 3.4.1 (Closest pair): Given n points in the plane, find two points that are closest.

It is obvious that the generalization of the problem in k dimensions, $k \geq 1$, can be solved in $O(kn^2)$ time by comparing all interpoint distances. In one dimension, we can solve the problem easily in $O(n \log n)$ time by a preliminary sorting. It turns out that sorting does not generalize to higher dimensions. Using the divide-and-conquer technique, Bentley and Shamos [40] showed that $O(n \log n)$ time is sufficient to solve this problem in dimensions $k \geq 1$, and the time bound is optimal (see Section III-F). An average-case study of this problem is presented in [45] where also an optimal average-case algorithm is illustrated. See [93], [158], [189] for other related results.

Problem 3.4.2 (All nearest neighbors): Given n points in the plane, find for each point a nearest neighbor (other than itself).

Problem 3.4.3 (Euclidean minimum spanning tree, EMST): Given n points in the plane, find a tree that interconnects all the points with minimum total edge length.

This problem has an obvious application in computer networking where we want to interconnect all the computers at minimum cost. This formulation, however, forbids the addition of extra points. If additional points, called *Steiner points,* are allowed, the problem becomes the *minimal Steiner tree problem,* which has been shown to be NP-hard [160]. Note also that the EMST problem can be cast as a graph-theoretic problem, in which the weight of each edge is the distance between the two terminal vertices of the edge. In [90] several spanning tree algorithms have been illustrated. In general, the minimum spanning tree problem for a graph with n vertices requires $\Omega(n^2)$ time, for the minimum weight edge must be in the tree and there are $O(n^2)$ independent weights in the input; however, the Euclidean metric properties can be exploited so as to solve the EMST problem in $\theta(n \log n)$ time.

Problem 3.4.4 (Triangulation): Given n points in the plane, construct a planar graph on the set of points such that each face within their convex hull is a triangle.

This problem arises in numerical interpolation of bivariate data [208]–[210], [253], [289] where the function values are known at irregularly spaced points, and in the finite element method [67]. A triangulation of these n points can be used to approximate the function value at a new point as the interpolation of the function values at the vertices of the triangle containing the new point.

Problem 3.4.5 (Nearest neighbor search): Given n points in the plane, with preprocessing allowed, find the nearest neighbor of a query point.

This problem, also known as the "post office problem," arises in pattern classification [115] where the nearest neighbor decision rule is used to classify a new sample into the class to which its nearest neighbor belongs, and in information retrieval where the record that *best matches* the query record is retrieved [64], [156]. See, for example, [264] and [116] for other "distance" measures used in the nearest neighbor search problems.

Problem 3.4.6 (k Nearest neighbors search): The same as Problem 3.4.5, except that the k nearest neighbors are sought.

2) *The Voronoi Diagram:* The above problems can be solved by resorting to the *locus method* mentioned in Section II-C. Given a set S of n points $\{p_1, p_2, \cdots, p_n\}$, we shall compute the *Voronoi diagram* [300] of S, denoted Vor(S), which partitions the plane into n "equivalence"

classes, each of which corresponds to a point. Specifically, the equivalence class corresponding to p_i is the Voronoi polygon $V(p_i)$, which is formally defined as $V(p_i) = \{r \mid r \text{ in } R^2 \text{ and } d(r, p_i) \leq d(r, p_j),\ j \neq i\}$. In other words, $V(p_i)$ is the locus of points that are as close to p_i as any other point of S and can also be defined as the intersection of the half-planes $\cap_{i \neq j} H(p_i, p_j)$ where $H(p_i, p_j)$ is the half-plane determined by the perpendicular bisector of $\overline{p_i p_j}$ and containing p_i. Thus, the Voronoi diagram of a set of n points is just a collection of n Voronoi (convex) polygons, one for each point. The diagram is also called *Thiessen polygons* [59]. The following is a catalog of properties of the Voronoi diagram (see, e.g., [211], [212], [316]), which can be derived under the simplifying assumption that no four points of the given set are cocircular. (See Fig. 5 for a Voronoi diagram of a set of 16 points.) i) Every vertex, called *Voronoi point*, of the Voronoi diagram has degree three. ii) Every nearest neighbor p_j of each point p_i defines an edge of $V(p_i)$, which is a portion of the perpendicular bisector of $\overline{p_i p_j}$. iii) $V(p_i)$ is an unbounded polygon if and only if the point p_i is on the convex hull of the set S. iv) The straight-line dual of the Voronoi diagram is a triangulation of S. The triangulation is also known as *Delaunay triangulation* and *Dirichlet tessellation* [55]. v) The numbers of edges and vertices in the Voronoi diagram Vor(S) are both $O(n)$ where $n = |S|$.

Construction of the Voronoi Diagram: The Voronoi diagram of a set S of n points in the plane can be constructed in $O(n \log n)$ time using the divide-and-conquer technique. We first divide set S into two halves S_1 and S_2, separated by a vertical line L, with S_1 lying to the left of L and S_2 to the right. We then recursively construct the Voronoi diagrams for the two subsets of points and complete the computation by merging these two diagrams into the final Voronoi diagram. With property iii) one can derive that each unbounded edge of the Voronoi diagram belongs to the perpendicular bisector of a convex hull edge. Since the convex hulls of S_1 and S_2 are disjoint, their two common supporting lines (see Problem 3.1.3) correspond, respectively, to two unbounded edges of the Voronoi diagram Vor(S). These two edges belong to a polygonal line σ, called the *separating chain*, which plays a crucial role in the merge step. Indeed, σ partitions the plane into two (unbounded) regions, R_1 and R_2, lying, respectively, to the left and to the right of σ. It can be shown that $\text{Vor}(S) = (\text{Vor}(S_1) \cap R_1) \cup (\text{Vor}(S_2) \cap R_2)$, thereby providing a merging technique. The construction of σ is done edge-by-edge, starting from one of the two unbounded edges. In this process, the property that each Voronoi polygon is convex is used crucially to show that each edge of $\text{Vor}(S_1) \cup \text{Vor}(S_2)$ is inspected at most a fixed number of times, thereby obtaining a merge algorithm running in time proportional to their total number of edges, i.e., in time $O(|S_1| + |S_2|) = O(n)$ [196], [211], [212]. Therefore, the total time for constructing the Voronoi diagram for a set S of n points is $O(n \log n)$, which is optimal (see Section III-F).

Brown has demonstrated an interesting linkage between Voronoi diagrams and convex hulls. In [61], [62] he presented an alternative $O(n \log n)$ algorithm for constructing the Voronoi diagram, by transforming the problem of constructing a planar Voronoi diagram for an n-point set to the construction of the convex hull of n points in 3-dimensional space via a geometric transformation known as *inversion*. The technique is general, i.e., the Voronoi diagram in k-dimensional space can be obtained from a convex hull in $(k + 1)$-dimensional space (see [61] for more details).

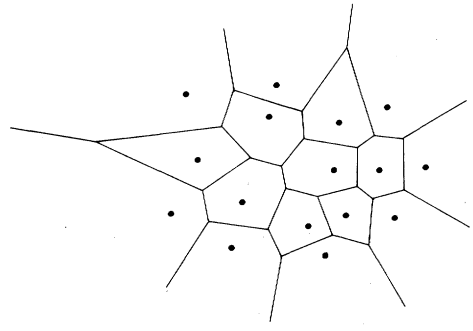

Fig. 5. The Voronoi diagram for a set of 16 points.

Once the Voronoi diagram is available, the closest-pair problem, the all-nearest-neighbor-problem, and the triangulation problem can all be solved in $O(n)$ time as follows. Obtain the straight-line dual graph by scanning each edge of the Voronoi diagram; since the dual graph is a triangulation and the total number of edges in Vor(S) is $O(n)$ [properties iv) and v)], the process takes $O(n)$ time. From property ii) we also have that the closest pair is identified with an edge of the triangulation and, similarly, the nearest neighbor of each point is given by an edge of the triangulation; therefore, both problems can be solved in $O(n)$ time. It has been shown [316] that the EMST is a subgraph of the Delaunay triangulation. So the EMST problem can also be solved in additional $O(n)$ time using the algorithm of Cheriton and Tarjan [90]. As for the nearest neighbor search problem, all we need to do is to find the Voronoi polygon in which the new point lies. The search is therefore a *point-location problem*, as discussed in Section III-C, and can be carried out in $O(\log n)$ time (cf. Problem 3.3.10).

Extensions of the Voronoi Diagram: There has been a number of extensions and generalizations of the Voronoi diagram. The Voronoi diagram discussed above refers to the Euclidean metric (i.e., the so-called L_2-metric in Minkowski's formulation). The definition of the Voronoi diagram can be easily extended to the L_p-metric where $1 \leq p \leq \infty$ [212], and the diagram can still be constructed in $O(n \log n)$ time, if we allow that the computation of the pth root can be done in constant time. The notion of spanning tree can also be extended to the L_p-metric [177], [232]. We note that the Voronoi diagram under the L_1-metric is not unique, and as a consequence, its dual graph is *not* a Delaunay triangulation. The Delaunay triangulation is a (unique) triangulation such that the circumcircle of each triangle does not contain any other point in its interior. Thus, to compute the Delaunay triangulation under the L_1-metric we cannot rely on the Voronoi diagram and a direct approach is needed. An $O(n \log n)$ algorithm that computes the Delaunay

triangulation directly is given in [230].

The second extension consists of considering the Voronoi diagram of a set of objects other than points. In [211], [222] the Voronoi diagram for a set of line segments or circles is considered and an $O(n \log^2 n)$ time algorithm is given for its construction. The time bound was later improved by Kirkpatrick to $O(n \log n)$ [196]. The *medial axis* [115] of a simple polygon, known as the *skeleton,* turns out to be part of the Voronoi diagram of the simple polygon [217], [290]. The Voronoi diagram of line segments or circles under the L_2-metric consists of edges that are not, in general, straight-line segments. (They may include arcs of either parabolas or hyperbolas.) However, a recent result [183] (and independently, [16]) shows that the Voronoi diagram of a set of n circles in the *Laguerre* geometry consists of straight-line edges and is computable in $O(n \log n)$ time. This particular diagram is useful for computing the connected components of the set of disks, for computing the contour of the union of disks, and for testing if a point lies inside the union or not.

The third extension focuses on the fact that in the Voronoi diagram discussed so far each polygon is the locus of points *nearest* to one point. To be more precise the diagram should be termed the *nearest neighbor Voronoi diagram.* Shamos and Hoey [316] considered the order-k Voronoi diagram of a set of points where each polygon of the diagram is associated with k points, $k \geq 1$, with the property that for any point inside the polygon its k nearest neighbors are precisely the associated k points. With the order-k diagram the k-nearest neighbors search problem can be solved in $O(\log n + k)$ time; in this expression the first term accounts for point location and the second term for reporting the answer. Properties and a method for the construction of the order-k Voronoi diagram can be found in [133], [216], [316]. At the other end of the spectrum (from the nearest neighbor Voronoi diagram) is the farthest neighbor Voronoi diagram, which is actually the order-$(n-1)$ Voronoi diagram. The farthest neighbor Voronoi diagram for n points can be constructed in $O(n \log n)$ time [213], [315] and can be used to solve the diameter problem (see below) and the 1-center problem in $O(n)$ time.

The fourth extension is to associate each point with a positive weight, resulting in a "weighted" Voronoi diagram [53]. The weighted Voronoi diagram consists of n "regions," each of which is the locus of points whose weighted distance to a given point is minimum. In [17] it is shown that the "regions" associated with each point may not be connected and in fact, there can be $\theta(n^2)$ edges and vertices in the diagram. An $O(n^2)$ algorithm for constructing such a weighted diagram can be found in [17].

The last extension consists of generalizing the diagram or triangulation to higher dimensions. Some results in this regard can be found in [20], [51], [60], [200], [311], [343]. With the connection between the d-dimensional Voronoi diagrams and the $(d+1)$-dimensional convex hulls [61], and the convex hull algorithm given by Seidel [310], an efficient solution to the construction of the Voronoi diagrams in higher dimensions can be obtained. The minimum spanning tree problem in higher dimensions has also been addressed (see [33], [98], [158], [192], [267], [351]), but remains an outstanding subject for research.

3) Other Proximity Problems:

Problem 3.4.7 (Closest pair for convex polygons): Given a convex polygon in the plane, find the two closest vertices.

Problem 3.4.8 (All nearest neighbors for convex polygons): Given a convex polygon in the plane, find the nearest neighbor for each vertex.

Exploiting the convexity of the polygon, Lee and Preparata have provided a linear time algorithm [226], [348] for these two problems. However, whether or not the closest pair of vertices of a simple polygon with n vertices can be found in $o(n \log n)$ time remains an open question. The known lower-bound proof (cf. Lemma 3.6.5) does not apply to this case since the vertices are given in a known order.

Problem 3.4.9 (Farthest pair or diameter problem): Given n points in the plane, find the two that are the farthest apart.

This problem arises in clustering [175] and image processing [322]. Since the farthest pair of points must be on the convex hull, the problem can be solved in $O(n)$ time in 1-dimension and in $O(n \log n)$ time in 2-dimension [295], [313], both results being optimal (see Section III-F); the 2-dimensional method consists of computing in $O(n \log n)$ time the convex hull of the n points, and then searching in $O(n)$ time for the two farthest points with the rotating calipers as given in [316], [330]. Note that the closest pair problem requires $\Omega(n \log n)$ time in all dimensions $d \geq 1$. For the farthest pair problem in higher dimensions, $d \geq 3$, Yao [351] has obtained an $o(n^2)$ algorithm. (Specifically, the time bound is $T(n,d) = O(n^{2-a(d)}(\log n)^{1-a(d)})$ where $a(d) = 2^{-(d+1)}$. For $d = 3$ the time bound can be improved to $O((n \log n)^{1.8})$.) Whether or not the gap between $T(n,d)$ and $\Omega(n \log n)$ can be reduced is an open question.

Problem 3.4.10 (Diameter problem for simple polygon): Given a simple polygon with n vertices, find the two that are farthest apart.

This problem can be solved in $O(n)$ time since the convex hull of the simple polygon can be constructed in $O(n)$ time and the farthest pair of a convex polygon can be found in linear time. Thus, convexity plays an important role here as well. But whether it can also help in 3 dimensions remains to be seen. Specifically, there is no known algorithm for finding the diameter of a convex polyhedron in less than $O((n \log n)^{1.8})$ time [351]. Note that the farthest pair problem for simple polygons is simpler (at least no harder) than the closest pair problem for simple polygons.

The following problems are of somewhat different type in that they deal with the distance between two *point sets*. The distance between two sets A and B of points is normally defined as the minimum distance between a point in A and a point in B, namely, $d(A,B) = \min_a \min_b d(a,b)$ where a in A, b in B, and d are the Euclidean distance. With this definition $d(A,B) = d(B,A)$. Another measure that is of interest is the *Hausdorff* distance [170], which is not symmetric. The *Hausdorff* distance from A to B is $\max_a \min_b d(a,b)$, and the Hausdorff distance between two sets A and B is equal to $\max\{d(A,B), d(B,A)\}$.

Problem 3.4.11 (Closest pair between sets): Given two sets A and B (with m and n points, respectively, if A and B are finite) find two points, one from each set, that are closest.

If A and B are both finite, this problem can be solved in $O(N \log N)$ time where $N = m + n$, by means of the Voronoi diagram. That is, each point a in A is located in Vor(B) and each b in B is located in Vor(A), resulting in total time $O(m \log n + n \log m)$ time after both diagrams have been computed. Note that the time bound is optimal (cf. Lemma 3.6.11).

When the points are the vertices either of a simple polygon or of a convex polygon, faster solutions are expected. Reference [14] discusses the Hausdorff distance between two convex polygons and gives an $O(n)$ time algorithm; [49] gives an $O(n \log n)$ algorithm for computing the *farthest* pair between two sets and an $O(n)$ algorithm for the same problem when the point sets form each a convex polygon. Reference [308] considers the problem of finding the closest pair of points between two convex polygons (two nonfinite sets), and gives an $O(\log^2 n)$ time algorithm; this result was later improved to $O(\log n)$ [91], [127]. To find the closest pair of vertices of two convex polygons, $O(n)$ time is both sufficient and necessary [92], [252], [331].

Problem 3.4.12 (Fixed-radius near neighbor): Given a constant $d > 0$ and n points in the plane, find for each point the neighbors within the distance d.

This problem arises in molecular graphics, cluster analysis, and data transmission [44]. Using the locus method, Bentley *et al.* [44] have shown that this problem under the L_∞-metric and sparsity condition, i.e., no more than a constant number c of points are in the hypersphere of diameter d, can be solved in $O(3^k kn(\log n + c))$ time where k is the dimension. When the sparsity condition is removed, they show that the total number of distance calculations is $3^k F$ where F is the total number of near neighbor pairs, i.e., the output size.

Problem 3.4.13 (Shortest path problem with obstacles): Given n geometric objects in the plane (called "obstacles") and two designated points s and t, find a shortest path between s and t which does not cross the interior of any obstacle.

Two instances of this problem have been studied in [229], both solvable in time $O(n \log n)$. In the first case the obstacles are the boundary edges of a simple polygon containing both s and t; in the second case, s and t are arbitrary points in the plane and the obstacles are a collection of parallel segments. In [224] an $O(n^2 \log n)$ algorithm for this problem in which the obstacles are n disks is presented. Tompa [327] examines this problem in the context of wire routing. See [206], [243], [309], [318] for other results along this line.

4) Polygon Decompositions Problems: Polygon decomposition problems have applications to pattern recognition. To facilitate the process of recognizing a shape a frequent strategy is to decompose the shape into simple parts ("primitives") and compare them to the library entries via some similarity measure [274]. Often the primitives are restricted to some particular types, such as convex polygons, star-shaped polygons, etc. In [328], Toussaint calls this class of decomposition *component-directed* (see [328], [329] for more references). Notice also that "triangulation" is a problem in this class. There are situations in which certain calculations are difficult for general polygons but easy for convex polygons; in this case, it is to our advantage to decompose a general simple polygon into convex parts and perform computations on each part. This is the approach taken by Ahuja *et al.* [2] for the detection of interference or collision of polygons.

Basically, there are two types of decompositions: *partition,* which disallows overlapping of component parts, and *covering,* which does allow overlapping parts. Sometimes additional vertices, called *Steiner points,* may be introduced to obtain decompositions with the minimum number of parts. A recent survey by Keil and Sack [194] discusses minimal decompositions in great detail.

Problem 3.4.14 (Triangulation of a simple polygon): Given a simple polygon P with n edges, decompose its interior into triangles.

A pioneering work is due to Garey *et al.* [162] and runs in $O(n \log n)$ time. The algorithm consists of two phases. In the first phase the polygon P is partitioned into a number of *monotone* polygons in $O(n \log n)$ time. (A polygon Q is *monotone* if there exists a line L such that the boundary of Q can be divided into two chains of edges and each chain is intersected at most once by any line orthogonal to L.) The second phase is a *linear* time algorithm for partitioning each monotone polygon into triangles. Other $O(n \log n)$ time triangulation algorithms are reported in [70], [334]. Recently, Chazelle and Incerpi have introduced a useful notion, called *sinuosity* of P [88], which is the number of times the boundary of P alternates between complete spirals of opposite directions, and obtained a divide-and-conquer algorithm for triangulating a simple polygon P in time $O(n \log s)$ where s is the sinuosity of P. Whether or not triangulating a simple polygon can be done in $O(n)$ time remains an open problem. If holes are allowed, then it has been shown that $\Omega(n \log n)$ time is necessary [12].

Problem 3.4.15 (Triangulation of a star-shaped polygon): Given a star-shaped polygon with n edges, decompose its interior into triangles.

Schoone and van Leeuwen [307] give an $O(n)$ time algorithm for this problem. In fact, more generally, any L-convex polygon can be triangulated in linear time [144].

Problem 3.4.16 (Quadrilaterization of a rectilinear polygon): Given a rectilinear polygon with n edges, decompose its interior into convex quadrilaterals.

A simple polygon does not always admit a quadrilaterization, but a rectilinear polygon always does [188]. Sack [302] has shown that any rectilinear polygon with n edges can be partitioned into convex quadrilaterals in $O(n \log n)$ time. It is an open problem if $o(n \log n)$ time is possible. If the rectilinear polygon is *monotone* or *star shaped,* then its quadrilaterization can be found in linear time [303].

Problem 3.4.17 (Decomposition of a simple polygon into star-shaped polygons): Given a simple polygon P with n edges, decompose its interior into star-shaped polygons.

Avis and Toussaint give an $O(n \log n)$ time for this problem [23]. They first find a triangulation of P and then perform a 3-coloring of the vertices of P so that no adjacent

vertices in the triangulation receive the same color. Based on the coloring, they remove all diagonals incident upon vertices of a given color, thereby obtaining a star-shaped partition. If a minimal partition is sought, Keil [193], using a dynamic programming technique, gives an $O(n^5N^2 \log n)$ algorithm where N denotes the number of *reflex* vertices (whose internal angle is greater than 180°).

Problem 3.4.18 (Decomposition of a simple polygon into convex parts): Given a simple polygon P with n edges, find a decomposition of its interior into convex polygons.

If Steiner points are allowed, Chazelle [69] gives an $O(n + N^2 \log(n/N))$ time algorithm where N denotes the number of *reflex* vertices. Chazelle and Dobkin [81] have also presented an $O(n^6)$ algorithm for finding a *minimal* partition of P into convex parts. The result was later improved to $O(n + N^3)$ [69]. In 3 dimensions an $O(nN^3)$ algorithm is presented in [79] and the algorithm produces $O(N^2)$ convex parts, which is shown to be worst-case optimal. Asano and Asano [9] have obtained an $O(n^3)$ time algorithm for partitioning P into a *minimum* number of *trapezoids* with two horizontal sides. The bound has been improved to $O(n^2)$ [10]. If no Steiner points are allowed, Feng and Pavlidis [145] describe an $O(nN^3)$ algorithm and Greene [169] gives an $O(n \log n)$ algorithm for partitioning P into convex parts (not necessarily minimum). However, if a minimal partition is sought, there are two algorithms reported with running times $O(n^2N^2)$ [169] and $O(N^2n \log n)$ [193].

There are decompositions with other objective functions. For example, the problem of finding in polynomial time a minimum weight triangulation, i.e., a triangulation with minimum total edge length of a set of n points in the plane, is still an open problem [161]. But interestingly enough, the minimum weight triangulation of a simple polygon can be done in $O(n^3)$ time [163], [202]. For results along this line the reader is referred to [193] and to the survey [194].

Problem 3.4.19 (Decomposition of rectilinear polygon into rectangles): Given a rectilinear polygon RP with n sides, decompose its interior into rectangles.

The minimal partition problem is elegantly solved by Lipski [237] in $O(n^{3/2} \log n \log \log n)$ time, with Steiner points allowed. Independently, Imai and Asano have solved the same problem in $O(n^{3/2} \log n)$ time [180], [181]. Both algorithms make use of maximum matching of a bipartite intersection graph of a set of vertical and horizontal line segments and require $O(n \log n)$ space.

Problem 3.4.20 (Decomposition of polygons with holes): Given a simple polygon P with "holes" in its interior, decompose its interior into simpler parts. (A *hole* is itself a simple polygon and cannot contain holes.)

It turns out that polygons with holes are much more difficult to decompose into simpler components, such as convex, star-shaped, or monotone parts. When Steiner points are allowed, partitioning a simple polygon with holes into a minimal number of triangles or convex parts [236] or into a minimal number of trapezoids with two horizontal sides [9] are both NP-hard problems. An $O(n^{2+h})$ algorithm [10] has been obtained to partition a simple polygon with n vertices and h holes into a minimal number of trapezoids with two horizontal sides. With Steiner points, the minimal covering of a simple polygon with convex parts, star-shaped, or spiral components is NP-hard [274]. We note that neither minimal covering nor minimal partition is known to be in NP. However, both problems have been shown to be decidable [72], [271]. Without Steiner points, both minimal covering and partitioning of a simple polygon with holes into convex, star-shaped, or spiral components are NP-hard [193], [236], [274].

Problem 3.4.21 (Decomposition of rectilinear polygons with holes into rectangles): Given a rectilinear polygon with rectilinear holes, decompose its interior into rectangles.

The minimal covering version of this problem is NP-hard [247]. However, the problem of finding the minimal partition is solvable in $O(n^{5/2})$ time [238], [270] by means of maximum bipartite matching techniques. In the paper [181] Imai and Asano give an $O(n^{3/2} \log n)$ time algorithm for this problem. If the holes can degenerate to points, the minimal partition problem becomes NP-hard [236].

Since most of the minimal decomposition or partition problems are presently intractable, good heuristics are therefore of interest. Only a few results with known performance bounds are available [9]–[11].

Problem 3.4.22 (Delaunay decomposition of simple polygons): Given a simple polygon P with n vertices, find a decomposition based on the Delaunay triangulation, i.e., with the property that the circumcircle of each triangle does not contain any other vertex of P *visible* from any of the three vertices of the triangle.

This is a generalization of the Delaunay triangulation of a planar point set discussed in Section III-D-2. In [211] an $O(n^2)$ algorithm is given, based on the notion of a *visibility graph* $VG(P)$ of the set of vertices of P. The graph $VG(P)$ has the same vertex set as P and two vertices are connected by an edge if the line segment determined by the two vertices does not intersect the boundary of P.

E. Geometric Optimization Problems

This section covers geometric problems of a combinatorial nature that are also frequently considered in the context of graph theory or optimization. The Euclidean minimum spanning tree problem (Problem 3.4.3), for example, is one such instance whose graph-theoretic counterpart is better known. As computational geometry evolved, the metric instances of such optimization problems were studied, with the objective to obtain more efficient algorithms. However, there are problems whose geometric instantiation does not seem to offer any advantage; the Euclidean traveling salesman problem and minimal Steiner tree problem are two notorious examples (they both remain NP-hard [160], [284]). The following is a list of geometric optimization problems that have received considerable attention in the past decade. Obviously, a large number of optimization problems has not yet been addressed.

Problem 3.5.1 (Linear programming): Given n half-spaces in d dimensions and a vector $\boldsymbol{x} = (x_1, x_2, \cdots, x_d)$, find a point $v = (v_1, v_2, \cdots, v_d)$ that belongs to the intersection of the half-spaces such that $x_1v_1 + x_2v_2 + \ldots + x_dv_d$ is maximized.

This is one of the best known problems in operations research and has a long history [96]. This problem in two

dimensions can be easily solved in $O(n \log n)$ time [313], [317] by finding the intersection of n half-planes (Problem 3.2.7). An optimal $O(n)$ time solution to this problem is represented by the elegant method of Dyer [118] and Megiddo [257], which we have described in some detail in Section II-F. The problem in higher dimensions can also be solved in $O(n)$ time when the dimensionality is fixed [259]. Dobkin *et al.* [109] have shown that the linear programming problem is Log-space hard for P. Dobkin and Reiss [112] recently introduced the notion of LP-complete, i.e., a problem is said to be LP-complete if and only if it is polynomially transformable to the problem of linear programming and *vice versa,* and have described a class of LP-complete problems.

Problem 3.5.2 (2-Dimensional linear separability): Given two sets P and Q of points in the plane with m and n points, respectively, determine if there exists a line that separates the two sets.

This problem has applications in statistics and pattern recognition [115]. Observing that P and Q are linearly separable if and only if the intersection of the convex hulls of P and Q is empty, Shamos [313] gives an $O(N \log N)$ time algorithm where $N = m + n$ by computing the two convex hulls in two dimensions and testing if they intersect. The same approach can be used in three dimensions and the same bound holds [293]. If the two point sets form convex polyhedra, Muller and Preparata [265] and later Dobkin and Kirkpatrick [104] have obtained linear algorithms to find a plane separating the two convex polyhedra, if it exists. However, the technique of Dyer [118] and Megiddo [257] can be used to find the separating line and plane of any two sets P and Q in 2- and 3-dimensions in $O(N)$ time. In general, a separating hyperplane can be found in $O(N)$ time for any fixed number of dimensions [259]. A recursive method for finding a separating hyperplane is given in [187].

Problem 3.5.3 (Smallest enclosing circle): Given n points in the plane, find the smallest circle that encloses all the points.

This problem is known as the 1-center problem in location theory [148]. The problem can be solved easily in $O(n \log n)$ time [316] by means of the farthest neighbor Voronoi diagram of these n points (cf. Section III-D). Using the prune-and-search approach, Megiddo [257] has obtained an $O(n)$ time algorithm for finding the smallest enclosing hypersphere.

Problem 3.5.4 (Smallest enclosing rectangle): Given a set of n points in the plane, find the smallest (area) rectangle enclosing the set.

This problem can be solved in $O(n \log n)$ time and is based on the observation, due to Freeman and Shapira [154], that the minimum rectangle must have a side parallel to an edge of the convex hull of S. After the convex hull is constructed in $O(n \log n)$ time, the minimum rectangle can be found in $O(n)$ time using the caliper method [313], [330]. No $o(n \log n)$ time algorithm is known for this problem. We note that this problem is related to the *Euclidean 1-line center problem,* i.e., given a set S of n points in the plane, find a straight line L such that the maximum Euclidean distance from the points in $S-L$ is minimized; the similarity resides in the fact that the line L must be parallel to an edge of the convex hull of S. It is shown in [235] that $O(n \log n)$ time is both necessary and sufficient for the 1-line center problem.

Problem 3.5.5 (Smallest enclosing triangle): Given a set S of n points in the plane, find the smallest (area) triangle enclosing the set.

Klee and Laskowski [201] recently studied the problem of finding all local minima among all the triangles enclosing a convex polygon with n vertices and gave an $O(n \log^2 n)$ time algorithm. (A triangle T is a *local minimum* if there exists a $c > 0$ such that the area of T' is no less than that of T for each enclosing triangle T' at Hausdorff distance (cf. Section III-D) less than c from T.) An optimal $\theta(n)$ algorithm is later given by O'Rourke *et al.* [272] for finding the smallest enclosing triangle for convex polygons with n vertices.

Problem 3.5.6 (Largest empty circle): Given a set S of n points in the plane, find the largest circle whose center lies within the convex hull of S and which does not contain any point of S in its interior.

This problem has an application in facility location where one wishes to place, for example, a dump site in a residential area so that the minimum distance from the site to a household is maximized. This problem can be solved in optimal time $\theta(n \log n)$ using the Voronoi diagram of S [313], [332]. The largest empty (isothetic) square problem can also be solved in $\theta(n \log n)$ time using the Voronoi diagram in L_∞-metric [212], [232].

Problem 3.5.7 (Largest empty rectangle): Given a set S of n points within a rectangle, find the largest (area) rectangle similar to the given rectangle that does not contain any point of S in its interior.

This problem is first studied in [266] where an $O(n^2)$ worst case time and an $O(n \log^2 n)$ expected time algorithms are given. In [83] the worst case time has been improved to $O(n \log^3 n)$ by means of a divide-and-conquer algorithm and of a modified version of the Voronoi diagram. The problem of finding a largest arbitrarily oriented rectangle seems much more difficult.

There are other results for similar geometric optimization problems. Dobkin *et al.* [100] discuss, among others, the smallest perimeter k-gon problem, i.e., the problem of finding a polygon with k vertices chosen from the given set of points whose perimeter is minimum, and show that the problem is NP-hard when k is $\Omega(n^c)$. For fixed k, the problem can be solved in $O(n \log n)$ time. Note that when k equals 2, the problem becomes the closest pair problem (Problem 3.4.1). In [56] an $O(n \log n)$ time algorithm for finding the maximum perimeter triangle, and an $O(kn \log n + n \log^2 n)$ algorithm for finding the maximum perimeter or area k-gon, for general k, are given. The largest area triangle and quadrilateral with vertices chosen from those of a convex polygon can be found in linear time [113], [315]; the smallest area triangle with vertices chosen from a given set of n points can be found in $O(n^2 \log^2 n)$ time [110]; this has been later improved to $O(n^2)$ [86], [133]. Finding the smallest enclosing ellipse is the subject of recent work by Silverman and Titterington [319] and Post [287], [288].

Next we consider two general location problems — p-center problem and p-median problem — which are more commonly cast in graph-theoretic terms and have been shown to be NP-hard [190], [191]. The Euclidean p-center problem

is formally defined as follows. Given a set S of points $\{p_1, p_2, \cdots, p_n\}$ in the plane with w_i the weight of p_i, find a set F of p points, known as *centers,* such that the maximum "distance" between the sets S and F of points is minimized, i.e., minimize $\max_i \min_j w_i d(p_i, q_j)$ where p_i in S, q_j in F, and $d(f_i, q_j)$ is the Euclidean distance between points p_i and q_j. The p-median problem, on the other hand, seeks to minimize the sum, rather than the maximum, of distances between the sets S and F. The Euclidean p-center problem is a standard minimax problem and the smallest enclosing circle problem is just the unweighted 1-center problem. As has been observed in the past, to prove NP-hardness of a geometric problem is more complicated than of its graph-theoretic counterpart [160], [284]. Nevertheless, these two problems have recently been shown to be also NP-hard [260]. In fact, even an approximation solution to the Euclidean p-center problem remains NP-hard, and a similar statement holds for the rectilinear p-center problem [260]. The following problem is also NP-complete [260].

Problem 3.5.8 (Disk cover): Given n points in the plane and p disks of the same radius r, determine if there exists a placement of these p disks so as to *cover* all the points, i.e., any given point lies inside at least one disk.

Consequently, the problem of determining the minimum number of disks of the same radius r that can cover n given points is also NP-hard. Papadimitriou has shown that the Euclidean p-median problem, in which the set F is restricted to be a subset of S, is NP-hard [285]. We note that the p-center problem in one dimension can be solved in $O(n \log n)$ time [149] and the p-median problem can be solved in $O(n^2 p)$ time [261].

We conclude this section with some variants of the above location problems. The *smallest bomb problem* [315], [325], i.e., find the smallest disk that covers at least k of the n given points, can be solved in time $O(k^2 n \log n)$ [216]; an approximation algorithm for this problem, generalizable to higher dimensions, is given in [325]. The *fixed-size bomb placement* problem, i.e., find a placement of the given disk in the plane so that the number (or the total weight) of points covered by the disk is maximized, can be solved in $O(n^2)$ time [89], which represents an improvement over an earlier algorithm [114] running in $O(n^2 \log n)$ time. If disks are replaced by rectangles, then the *fixed-size rectangle placement problem* can be solved in optimal time $\theta(n \log n)$ in two dimensions [179] and in $O(n^{d-1})$ time in d dimensions, $d > 2$ [219]. The weighted 1-center problem in the plane can be solved in $O(n \log^2 n)$ time [255]. More efficient algorithms can be obtained in the L_1-metric [256]. Cole [94] has recently shown that the weighted 1-center problem can be solved in $O(n \log n)$ time. An interesting problem would be whether or not this is the best possible for the problem. (Recall that the unweighted 1-center problem can be solved in $O(n)$ time [257].) Other combinatorial optimization problems, such as matching of $2n$ points in the plane, have also been studied. A survey done by Avis [19] discusses heuristics for the matching problems in some detail.

F. Problem Transformations and Lower Bounds

In this section we summarize a few complexity results that have been established either by a direct argument on the height of the computation/decision tree, or by problem transformations (to be defined below). We first state a few basic results without proofs.

Fact 1: (Membership) Given a set S of n objects, to determine if a particular object belongs to S requires at least $\lceil \log_2 n \rceil$ tests in the worst case. This is the so-called *information-theoretic bound* for any membership problem [54].

Fact 2: (Sorting) Given n elements from a totally ordered set, the problem of sorting these n elements in order requires at least $\Omega(n \log n)$ comparisons under any comparison-based or decision-tree model of computation (DTM for short) [1]. This bound can be viewed as an information-theoretic bound since we look for a particular permutation in a set of $n!$ possible permutations of these n elements and $\lceil \log_2 n! \rceil = \Omega(n \log n)$.

Fact 3: (Dimension embedding) If a problem in d dimensions requires $f(n)$ time, then the problem in k dimensions, $k > d$, also requires $f(n)$ time (see, e.g. [18]).

Definition: Given two problems A and B, A is said to be *$f(n)$-transformable* to B, if any instance of A can be transformed into an instance of B and the solution to the instance of B can be transformed back to a solution to the instance of A within $O(f(n))$ time where n is the size of problem A.

Lemma 3.6.1 [315]: Suppose problem A is $g(n)$-transformable to problem B. If A is known to require $f(n)$ time under a certain model of computation, then B requires at least $f(n) - cg(n)$ time, for some constant $c > 0$. If B is solvable in $f(n)$ time, then A is solvable in $f(n) + c'g(n)$ time, for some constant $c' > 0$.

As we mentioned before, we shall use the algebraic computation tree (ACT for short) of Ben-Or [26] as our primary computation model, possibly augmented with a number of primitive operations such as testing on which side of a directed line a point lies, all assumed to be executable in constant time. A first-order ACT is referred to as a linear computation tree (LCT). The following problems can be shown to require $\Omega(n \log n)$ time under the ACT model of computation [26].

Element Uniqueness Problem [108]: Given n numbers, determine if they are all distinct.

Set Equality and Inclusion Problem [298]: Given two sets $A = \{x_1, x_2, \cdots, x_n\}$ and $B = \{y_1, y_2, \cdots, y_n\}$, determine if $A = B$ or $A \subseteq B$.

Set Disjointness Problem [298]: Given two sets $A = \{x_1, x_2, \cdots, x_n\}$ and $B = \{y_1, y_2, \cdots, y_n\}$, determine if $A \cap B = \emptyset$.

Extreme Points Problem [324], [350]: Given n points in the plane, does their convex hull possess n vertices?

ε-Closeness Problem [150]: Given n real numbers, determine if any two are within ε of each other where ε is a fixed parameter of the problem.

Measure Problem [150]: Given n intervals on the real line, compute the union of these n intervals. (This follows from the fact that ε-closeness problem is $O(n)$-transformable to it.)

Interval Compactness Problem [305]: Given n intervals on the real line, determine if the union of these n intervals is

itself an interval.

The following problem requires $\Omega(n \log n)$ time under the LCT model of computation.

Even Distribution Problem [245]: Given n numbers and a fixed parameter $c > 0$, determine if they are evenly distributed with spacing c. (A set of reals $\{x_1, x_2, \cdots, x_n\}$ is said to be *evenly distributed* with spacing $c > 0$, if there exists a permutation p of $\{1, 2, \cdots, n\}$ such that $x_{p(1)} \leq x_{p(2)} \leq \cdots \leq x_{p(n)}$ and $0 \leq x_{p(i+1)} - x_{p(i)} \leq c$, for $i = 1, 2, \cdots, n-1$.)

We now make use of Lemma 3.6.1 to sketch the proof of some additional lower bounds. Seidel [312] gives a few lower bound results for problems where certain additional geometric information possessed by the input does not lead to more efficient solutions.

Lemma 3.6.2 [315], [316]: The convex hull problem of n points in the plane requires $\Omega(n \log n)$ time in the ACT model.

Proof: By $O(n)$-transformation from sorting. The transformation is as follows. Given n positive numbers $x_1, x_2, \cdots, x_n$, we map each number to a point on the parabola $y = x^2$, i.e., x_i is mapped to the point $p_i = (x_i, x_i^2)$. Since the convex hull of $p_1, p_2, \cdots, p_n$ can be reported in order starting from the point with the smallest x-coordinate, any convex hull algorithm can sort [Fig. 6(a)].

Lemma 3.6.3 [317]: The half-plane intersection construction problem for n half-planes requires $\Omega(n \log n)$ time in the ACT model.

Proof: By $O(n)$-transformation from sorting. Given n real numbers $x_1, x_2, \cdots, x_n$, we map each number x_i to a half-plane H_i containing the origin and determined by the tangent to the parabola $y = x^2$ with slope $2x_i$, i.e., x_i is mapped to $y \geq 2x_i x - x_i^2$ [Fig. 6(b)]. Since the intersection of half-planes is a convex polygon whose successive edges are ordered by slope, any algorithm that constructs the intersection can sort.

Lemma 3.6.4 [317]: The intersection detection problem for n intervals on the real line requires $\Omega(n \log n)$ time in the ACT model.

Proof: By $O(n)$-transformation from element uniqueness. Indeed each number in the input of the element uniqueness problem can be considered as a degenerate interval.

Lemma 3.6.5 [316]: The closest pair problem for n points on the real line requires $\Omega(n \log n)$ time in the ACT model.

Proof: By $O(n)$-transformation from element uniqueness.

Even if the points of the problem are known to be distinct *a priori*, the problem still requires $\Omega(n \log n)$ time, by $O(n)$-transformation from the ε-closeness problem.

Lemma 3.6.6 [61]:[1] The diameter problem for n points in the plane requires $\Omega(n \log n)$ time in the ACT model.

Proof: By $O(n)$-transformation from set disjointness. Given two sets $A = \{x_1, x_2, \cdots, x_n\}$ and $B = \{y_1, y_2, \cdots, y_n\}$ where x_i and y_i are in the interval $(0, 1)$, we map these numbers onto the circumference of a unit circle centered at the origin as follows. First consider the set of points $S = \{p_i \mid p_i = (x_i, 1)\}$ and $T = \{q_i \mid q_i = (-y_i, -1)\}$, for $i = 1, 2, \cdots, n$. Next we apply the transform to map each point (x, y) in S and T into the point $(x/\sqrt{x^2 + y^2}, y/\sqrt{x^2 + y^2})$ on the unit circle (Fig. 7). This effectively maps a number in A onto a point in the first quadrant of the unit circle and a number in B onto a point in the third quadrant. Then it is immediate that the diameter of the set of points on the circle is 2 if and only if A and B are not disjoint.

[1] Dobkin and Munro have independently proved the lemma.

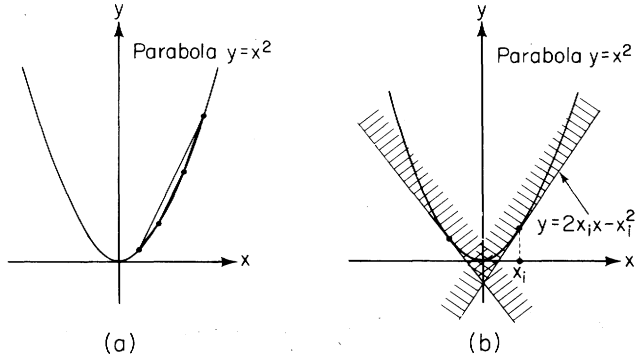

Fig. 6. Mappings of numbers x_i to points on parabola $y = x^2$ or to half-planes whose boundary lines are tangent to the parabola.

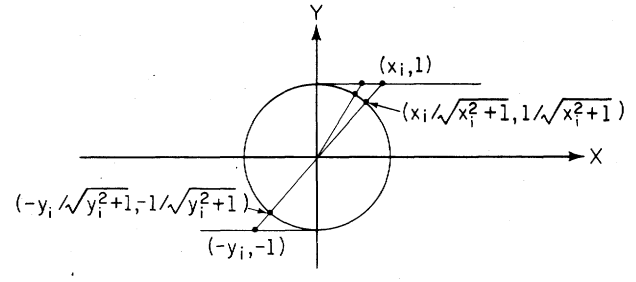

Fig. 7. Mapping of numbers onto the unit circle $x^2 + y^2 = 1$.

Lemma 3.6.7 [235]: The discrete 1-center problem for n points, i.e., given n points in the plane, find the point (the center) whose maximum distance to the remaining $n - 1$ points is minimized, requires $\Omega(n \log n)$ time in the ACT model.

Proof: By $O(n)$-transformation from set equality. Consider the mapping of two sets A and B of n numbers in $(0, 1)$, as given in the previous lemma. Since the distance realized by the center and its farthest neighbor is 2 if and only if the two sets A and B are equal, the claim follows.

Lemma 3.6.8 [313]: The Euclidean minimum spanning tree (EMST) problem for n points in the plane requires $\Omega(n \log n)$ in the ACT model.

Proof: By $O(n)$-transformation from sorting. Given n numbers $x_1, x_2, \cdots, x_n$ to be sorted, we map x_i to the point $p_i = (x_i, 0)$ on the x-axis, for $i = 1, 2, \cdots, n$. Reporting the EMST of these n points gives the ordered sequence of these points, whence any EMST algorithm can sort.

Remark: The closest pair problem is also $O(n)$-transformable to the EMST problem since the edge between the two closest points must belong to the tree. Hence, this provides an alternative proof of the $\Omega(n \log n)$ lower bound for the EMST problem.

Lemma 3.6.9 [313]: The triangulation problem of n points in the plane requires $\Omega(n \log n)$ time in the ACT model.

Proof: By $O(n)$-transformation from sorting. Given n numbers, we map them to n points on the x-axis as in Lemma 3.6.8. Adding to the set of n points a point not on the x-axis will yield a set whose triangulation can be reported so

as to give the ordered sequence of the x-coordinates of the points on the x-axis. Hence, any triangulation algorithm can sort.

Lemma 3.6.10 [205], [338]: The maxima problem for n points in the plane, i.e., finding all maxima of n points in the plane, requires $\Omega(n \log n)$ time in the DTM. (A point $p = (x, y)$ in the plane is a *maximum* if there exists no other point $q = (s, t)$ such that $x \leq s$ and $y \leq t$.)

Proof: By $O(n)$-transformation from sorting. Given n distinct numbers $x_1, x_2, \cdots, x_n$, we map them to points on the line L with equation $x + y = c$ for some constant c, i.e., x_i is mapped to point p_i on L with (x_i, y_i) as the x- and y-coordinates. Note that each point p_i is a maximum. To determine that p_i is a maximum the algorithm must be able to decide that there exists no point p_j, $j \neq i$, such that $x_i < x_j$ and $y_i < y_j$. This is equivalent to saying that for all $j \neq i$ either $x_i < x_j$ or $x_i > x_j$, that is, for each pair x_i and x_j the algorithm must have detected their ordering.

Lemma 3.6.11: The minimum distance between two linearly separable point sets of size n each requires $\Omega(n \log n)$ time in the ACT model.

Proof: By $O(n)$-transformation from set disjointness. Given two sets $A = \{a_1, a_2, \cdots, a_n\}$ and $B = \{b_1, b_2, \cdots, b_n\}$ where a_i and b_i are reals, we map these numbers to two linearly separable sets of points as follows. Let the numbers in A be mapped to points in set $P = \{(a_i, 0) \mid i = 1, 2, \cdots, n\}$ and let the numbers in B be mapped to points in set $Q = \{(b_i, 1) \mid i = 1, 2, \cdots, n\}$. It is easily seen that sets A and B have a common element if and only if the distance between sets P and Q is 1.

Lemma 3.6.12 [305]: The visibility problem of n horizontal line segments, i.e., given n horizontal line segments, determine if any two line segments are mutually visible, requires $\Omega(n \log n)$ in the ACT model. (Two horizontal line segments are said to be *mutually visible* if there exists a vertical line segment connecting two points of these segments without intersecting any other line segment.)

Proof: By $O(n)$-transformation from interval compactness. Given n intervals $I_1, I_2, \cdots, I_n$, we map each interval I_j to a horizontal line segment H_j of the same length at ordinate j. Next we add to this set of segments two new horizontal segments whose lengths are equal to the distance between the rightmost and the leftmost endpoints of the intervals and are situated at $y = 0$ and $y = n + 1$, respectively. Then these two new horizontal segments are mutually visible if and only if the union of the intervals is not an interval.

Lemma 3.6.13: The point-on-line problem,[2] i.e., given n points and n lines in the plane, determine if any point lies on any line, requires $\Omega(n \log n)$ time in the ACT model.

Proof: By $O(n)$-transformation from set disjointness. Given two sets $A = \{x_1, x_2, \cdots, x_n\}$ and $B = \{y_1, y_2, \cdots, y_n\}$ where x_i and y_i are in the interval $(0, 1)$, we map them on the unit circle as in Lemma 3.6.6. Then we convert the set of points corresponding to B into a set of n lines that pass through the origin and the points. It is obvious that the point-on-line problem has an answer YES if and only if the sets A and B are not disjoint.

Lemma 3.6.14: The collinearity problem, i.e., given n points in the plane, determine if any three are collinear, requires $\Omega(n \log n)$ time in the ACT model.

Proof: By $O(n)$-transformation from set disjointness. Given two sets A and B of n numbers in $(0, 1)$, we perform the mapping as in the previous lemma and add the origin to the set. The resulting set of $2n + 1$ points has three collinear points if and only if A and B are not disjoint.

Lemma 3.6.15 [245], [295]: The maximum gap problem, i.e., given n points on the real line, find the maximum gap defined by two consecutive points, requires $\Omega(n \log n)$ time in the LCT model.

Proof: By $O(n)$-transformation from even distribution. Indeed, we can compare the distance of the gap to the parameter c of the even distribution problem in constant time.

Remark: In our computation model the use of floor function as one of the primitive functions is ruled out. Note that Gonzalez has obtained an $O(n)$ time algorithm for this problem with the aid of the floor function ([165], see also [295]).

Lemma 3.6.16: The densest semicircle problem [186], i.e., given n points on a circle, find a closed semicircle containing the maximum number of these points, requires $\Omega(n \log n)$ time in the ACT model.

Proof: By $O(n)$-transformation from element uniqueness. Given n numbers $x_1, x_2, \cdots, x_n$, map x_i to the two points $(1/y_i, x_i/y_i)$, and $(-1/y_i, -x_i/y_i)$ where $y_i = \sqrt{1 + x_i^2}$. Then the numbers are distinct if and only if the densest closed semicircle contains exactly $n + 1$ points.

Lemma 3.6.17 [313], [315]: The construction of the Voronoi diagram of a set of n points in the plane requires $\Omega(n \log n)$ time in the ACT model.

Proof: We can transform a number of problems to it. Take, for example, the closest pair problem. Since the two closest points must define an edge in the Voronoi diagram, we can use the diagram to solve the closest pair problem.

Lemma 3.6.18: The largest empty circle problem in the plane, i.e., given n points, find the largest circle whose center is in the convex hull and which does not contain any point in its interior, requires $\Omega(n \log n)$ time in the LCT model.

Proof: This is just the maximum gap problem (3.6.15) in two dimensions.

IV. Conclusion

We have surveyed the state-of-the-art of a newly emerged discipline known as computational geometry. The survey is intended to be as broad as possible, although there are a number of research findings that are not included in this paper. The problem areas and the techniques are not outlined in their finest detail, but are described as accurately as possible.

There are several open problems, most of which have been mentioned in the article. The new notion of dynamic computational geometry [15], [275] where objects involved are moving over (continuous) time, for example, is one of the research directions that deserve further investigation. Presently, the primary emphasis in computational geometry is on the asymptotic performance of algorithms. In order to accelerate the already occurring technological transfer, it is recommended that increasing attention be paid to the nonasymptotic behavior of algorithms, i.e., to their performance for "small" problem sizes. Comparisons, such as [13], of the

[2] Hopcroft, private communication.

running times of various algorithms whose constants are hidden in the big-Oh notation (asymptotic analysis), for practical values of input size, are much needed.

ACKNOWLEDGMENT

The authors would like to thank Dr. C. K. Wong for his encouragement to write this survey and to express their deep gratitude to Profs. T. Asano, B. Chazelle, H. Edelsbrunner, and G. Toussaint for their comments and careful reading of an earlier draft of this survey.

REFERENCES

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms*, Reading, MA: Addison-Wesley, 1974.
[2] N. Ahuja, R. T. Chien, R. Yen, and N. Birdwell, "Interference detection and collision avoidance among three-dimensional objects," in *Proc. 1st Annu. Nat. Conf. Artificial Intell.*, Palo Alto, CA, 1980, pp. 44–48.
[3] S. G. Akl, "Two remarks on a convex hull algorithm," *Inform. Processing Lett.*, vol. 8, pp. 108–109, 1979.
[4] S. G. Akl and G. T. Toussaint, "A fast convex hull algorithm," *Inform. Processing Lett.*, vol. 7, pp. 219–222, 1978.
[5] ——, "Efficient convex hull algorithms for pattern recognition applications," in *Proc. 4th Int. Joint Conf. Pattern Recognition*, Kyoto, Japan, 1978, pp. 483–487.
[6] K. R. Anderson, "A reevaluation of an efficient algorithm for determining the convex hull of a finite planar set," *Inform. Processing Lett.*, vol. 7, pp. 53–55, 1978.
[7] A. M. Andrew, "Another efficient algorithm for convex hulls in two dimensions," *Inform. Processing Lett.*, vol. 9, pp. 216–219, 1979.
[8] A. Appel and P. M. Will, "Determining the three-dimensional convex hull of a polyhedron," *IBM J. Res. Develop.*, pp. 590–601, 1976.
[9] T. Asano and T. Asano, "Minimum partition of polygonal regions into trapezoids," in *Proc. 24th IEEE Annu. Symp. Foundations Comput. Sci.*, Nov. 1983, pp. 233–241.
[10] T. Asano, T. Asano, and H. Imai, "Partitioning a polygonal region into trapezoids," Dep. Math. Eng. Instrum. Phys., Univ. Tokyo, Tokyo, Japan, Res. Memo. RMI84-03, 1984.
[11] T. Asano, T. Asano, and Y. Ohsuga, "Partitioning polygonal regions into triangles," Papers of Tech. Groups of IECE, CAS 83-98, 1983, pp. 31–36.
[12] T. Asano, T. Asano, and R. Y. Pinter, "Polygon triangulation: Efficiency and minimality," submitted for publication, 1984.
[13] T. Asano, M. Edahiro, H. Imai, M. Iri, and K. Murota, "Practical use of bucketing techniques in computational geometry," in *Computational Geometry*, G. T. Toussaint, Ed. New York: North-Holland, to be published.
[14] M. J. Atallah, "A linear time algorithm for the Hausdorff distance between convex polygons," *Inform. Processing Lett.*, vol. 8, pp. 207–209, Nov. 1983.
[15] ——, "Dynamic computational geometry," in *Proc. 24th IEEE Annu. Symp. Foundations Comput. Sci.*, Nov. 1983, pp. 92–99.
[16] F. Aurenhammer, "Power diagrams: Properties, algorithms and applications," IIG, Tech. Univ. Graz, Graz, Austria, Rep. F120, 1983.
[17] F. Aurenhammer and H. Edelsbrunner, "An optimal algorithm for constructing the weighted Voronoi diagrams in the plane," *Pattern Recognition*, vol. 17, no. 2, pp. 251–257, 1984.
[18] D. Avis, "Lower bounds for geometric problems," in *Proc. 18th Allerton Conf. Commun., Contr., Comput.*, 1980, pp. 35–40.
[19] ——, "A survey of heuristics for the weighted matching problem," *Networks*, vol. 13, pp. 475–493, 1983.
[20] D. Avis and B. K. Bhattacharya, "Algorithms for computing the d-dimensional Voronoi diagrams and their duals," in *Advances in Computing Research*, Vol. 1, F. P. Preparata, Ed. JAI Press, 1983, pp. 159–180.
[21] D. Avis, H. El Gindy, and R. Seidel, "Simple on-line algorithms for convex polygons," in *Computational Geometry*, G. T. Toussaint, Ed. New York: North-Holland, to be published.
[22] D. Avis and G. T. Toussaint, "An optimal algorithm for determining the visibility of a polygon from an edge," *IEEE Trans. Comput.*, vol. C-30, pp. 910–914, Dec. 1981.
[23] ——, "An efficient algorithm for decomposing a polygon into star-shaped polygons," *Pattern Recognition*, vol. 13, pp. 395–398, 1981.
[24] H. S. Baird, "Fast algorithms for LSI artwork analysis," *J. Design Automat. Fault-Tolerant Computing*, vol. 2, no. 2, pp. 179–209, 1978.
[25] L. J. Bass and S. R. Schubert, "On finding the disc of minimum radius containing a given set of points," *Math. Comput.*, vol. 12, pp. 712–724, 1967.
[26] M. Ben-Or, "Lower bounds for algebraic computation trees," in *Proc. 15th ACM Annu. Symp. Theory Comput.*, Apr. 1983, pp. 80–86.
[27] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Commun. Ass. Comput. Mach.*, vol. 18, pp. 509–517, Sept. 1975.
[28] ——, "Solutions to Klee's rectangle problems," Carnegie-Mellon Univ., Pittsburgh, PA, 1977, unpublished.
[29] ——, "Decomposable searching problems," *Inform. Processing Lett.*, vol. 8, pp. 244–251, 1979.
[30] ——, "Multidimensional divide and conquer," *Commun. Ass. Comput. Mach.*, vol. 23, pp. 214–229, Apr. 1980.
[31] ——, "Notes on a taxonomy of planar convex hull algorithms," Dep. Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, manuscript, 1980.
[32] J. L. Bentley, G. M. Faust, and F. P. Preparata, "Approximation algorithms for convex hulls," *Commun. Ass. Comput. Mach.*, vol. 25, pp. 64–68, 1982.
[33] J. L. Bentley and J. H. Friedman, "Fast algorithms for constructing minimal spanning trees in coordinate spaces," *IEEE Trans. Comput.*, vol. C-27, pp. 97–105, Feb. 1978.
[34] ——, "Data structures for range searching," *Comput. Surveys*, vol. 11, pp. 397–409, 1979.
[35] J. L. Bentley, D. Haken, and R. Hon, "Fast geometric algorithms for VLSI tasks," in *Proc. Comput. Conf.*, 1981, pp. 88–92.
[36] J. L. Bentley and H. A. Maurer, "A note on Euclidean near neighbor searching in the plane," *Inform. Processing Lett.*, vol. 8, pp. 133–136, 1979.
[37] ——, "Efficient worst-case data structures for range searching," *Acta Inform.*, vol. 13, pp. 155–168, 1980.
[38] J. L. Bentley and T. Ottmann, "Algorithms for reporting and counting geometric intersections," *IEEE Trans. Comput.*, vol. C-28, pp. 643–647, Sept. 1979.
[39] J. L. Bentley and J. B. Saxe, "Decomposable searching problems, I. Static-to-dynamic transformation," *J. Algorithms*, vol. 1, pp. 301–358, 1980.
[40] J. L. Bentley and M. I. Shamos, "Divide-and-conquer in multidimensional space," in *Proc. 8th ACM Annu. Symp. Theory Comput.*, 1976, pp. 220–230.
[41] ——, "A problem in multivariate statistics: Algorithms, data structure and applications," in *Proc. 15th Allerton Conf. Commun., Contr., Comput.*, 1977, pp. 193–201.
[42] ——, "Divide and conquer for linear expected time," *Inform. Processing Lett.*, vol. 7, pp. 87–91, 1978.
[43] J. L. Bentley and D. F. Stanat, "Analysis of range searches in quad trees," *Inform. Processing Lett.*, vol. 3, pp. 170–173, 1975.
[44] J. L. Bentley, D. F. Stanat, and E. H. Williams, Jr., "The complexity of finding fixed-radius near neighbors," *Inform. Processing Lett.*, vol. 6, pp. 209–212, 1977.
[45] J. L. Bentley, B. Weide, and A. C. Yao, "Optimal expected time algorithms for closest-point problems," *ACM Trans. Math. Software*, vol. 6, no. 4, pp. 563–579, 1980.
[46] J. L. Bentley and D. Wood, "An optimal worst-case algorithm for reporting intersections of rectangles," *IEEE Trans. Comput.*, vol. C-29, pp. 571–577, July 1980.
[47] P. Bezier, *Numerical Control—Mathematics and Applications*, A. R. Forrest, Transl. New York: Wiley, 1972.
[48] B. K. Bhattacharya and H. El Gindy, "A new linear convex hull algorithm for simple polygons," *IEEE Trans. Inform. Theory*, vol. IT-30, pp. 85–88, Jan. 1984.
[49] B. K. Bhattacharya and G. T. Toussaint, "Efficient algorithms for computing maximum distance between two finite planar sets," *J. Algorithms*, vol. 4, pp. 121–126, June 1983.
[50] H. Bieri and W. Nef, "A recursive plane-sweep algorithm, determining all cells of a finite division of R^d," *Computing*, vol. 28, pp. 189–198, 1982.
[51] J. D. Boissonnat and O. D. Faugeras, "Triangulation of 3D objects," in *Proc. 7th Int. Joint Conf. Artificial Intell.*, Vancouver, B.C., Canada, 1981, pp. 658–660.
[52] A. Bolour, "Optimal retrieval algorithms for small region queries," *SIAM J. Comput.*, vol. 10, pp. 721–741, 1981.
[53] B. N. Boots, "Weighting Thiessen polygons," *Econ. Geogr.*, pp. 248–259, 1979.
[54] A. B. Borodin, "Computational complexity—Theory and practice," in *Currents in the Theory of Computing*, A. V. Aho, Ed. Englewood Cliffs, NJ: Prentice-Hall, 1973, pp. 35–89.

[55] A. Bowyer, "Computing Dirichlet tessellations," *Comput. J.*, vol. 24, pp. 162–166, 1981.
[56] J. E. Boyce, D. P. Dobkin, R. L. Drysdale, III, and L. J. Guibas, "Finding extremal polygons," in *Proc. 14th ACM Annu. Symp. Theory Comput.*, 1982, pp. 282–289.
[57] J. W. Boyse, "Interference detection among solids and surfaces," *Commun. Ass. Comput. Mach.*, vol. 21, pp. 3–9, 1979.
[58] K. E. Brassel and R. Fegeas, "An algorithm for shading of regions on vector display devices," *Comput. Graphics*, vol. 13, pp. 126–133, 1979.
[59] K. E. Brassel and D. Reif, "A procedure to generate Thiessen polygons," *Geogr. Anal.*, vol. 11, pp. 289–303, 1979.
[60] W. Brostow, J.-P. Dussault, and B. L. Fox, "Construction of Voronoi polyhedra," *J. Comput. Phys.*, vol. 29, pp. 81–92, 1978.
[61] K. Q. Brown, "Geometric transformations for fast geometric algorithms," Ph.D. dissertation, Dep. Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, Dec. 1979.
[62] ——, "Voronoi diagrams from convex hulls," *Inform. Processing Lett.*, vol. 9, pp. 223–228, 1979.
[63] ——, "Comments on 'Algorithms for reporting and counting geometric intersections,'" *IEEE Trans. Comput.*, vol. C-30, pp. 147–148, Feb. 1981.
[64] W. A. Burkhard and R. M. Keller, "Some approaches to best match file searching," *Commun. Ass. Comput. Mach.*, vol. 16, pp. 230–236, 1973.
[65] R. P. Burton and D. R. Smith, "A hidden-line algorithm for hyperspace," *SIAM J. Comput.*, vol. 11, pp. 71–80, 1982.
[66] A. Bykat, "Convex hull of a finite set of points in two dimensions," *Inform. Processing Lett.*, vol. 7, pp. 296–298, 1978.
[67] J. C. Cavendish, "Automatic triangulation of arbitrary planar domains for the finite element method," *Int. J. Numer. Methods Eng.*, vol. 8, pp. 679–696, 1974.
[68] D. R. Chand and S. S. Kapur, "An algorithm for convex polytopes," *J. ACM*, vol. 17, pp. 78–86, Jan. 1970.
[69] B. M. Chazelle, "Computational geometry and convexity," Ph.D. dissertation, Yale Univ., New Haven, CT; also, Carnegie-Mellon Univ., Pittsburgh, PA, Tech. Rep. CMU-CS-80-150, 1980.
[70] ——, "A theorem on polygon cutting with applications," in *Proc. 23rd IEEE Annu. Symp. Found. Comput. Sci.*, 1982, pp. 339–349.
[71] ——, "The polygon containment problem," in *Advances of Computing Research*, Vol. 1, F. P. Preparata, Ed. JAI Press, 1983, pp. 1–33.
[72] ——, "A decision procedure for optimal polyhedron partitioning," *Inform. Processing Lett.*, vol. 16, pp. 75–78, 1983.
[73] ——, "An improved algorithm for the fixed-radius neighbor problem," *Inform. Processing Lett.*, vol. 16, pp. 193–198, 1983.
[74] ——, "Filtering search: A new approach to query-answering," in *Proc. 24th IEEE Annu. Symp. Found. Comput. Sci.*, Nov. 1983, pp. 122–132.
[75] ——, "Optimal algorithms for computing depths and layers," in *Proc. 21st Allerton Conf. Commun. Control Comput.*, Oct. 1983, pp. 427–436.
[76] B. M. Chazelle, "Intersecting is easier than sorting," in *Proc. 16th ACM Annu. Symp. Theory Comput.*, 1984, pp. 125–134.
[77] ——, "How to search in history," in *Proc. Conf. Found. Comput. Theory*. New York: Springer-Verlag, Aug. 1983, pp. 52–63.
[78] ——, "Fast searching in a real algebraic manifold with applications to geometric complexity," Dep. Comput. Sci., Brown Univ., Providence, RI, Tech. Rep. TR-CS-84-13, June 1984.
[79] ——, "Convex partitions of polyhedra: A lower bound and worst-case optimal algorithm," *SIAM J. Comput.*, vol. 13, no. 3, pp. 488–507, Aug. 1984.
[80] B. M. Chazelle, R. Cole, F. P. Preparata, and C. K. Yap, "New upper bounds for neighbor searching," in preparation.
[81] B. M. Chazelle and D. P. Dobkin, "Decomposing a polygon into its convex parts," in *Proc. 11th ACM Annu. Symp. Theory Comput.*, 1979, pp. 38–48.
[82] ——, "Detection is easier than computation," in *Proc. 12th ACM Annu. Symp. Theory Comput.*, May 1980, pp. 146–153.
[83] B. M. Chazelle, R. L. Drysdale III, and D. T. Lee, "Computing the largest empty rectangle," in *Proc. Symp. Theoretic Aspects Comput. Sci.*, Paris, France, Apr. 1984.
[84] B. M. Chazelle and H. Edelsbrunner, "Optimal solutions for a class of point retrieval problems," Technische Univ. Graz, Austria, Tech. Rep. IIG, to be published.
[85] B. M. Chazelle and L. J. Guibas, "Fractional cascading: A data structuring technique with geometric applications," manuscript, 1984.
[86] B. M. Chazelle, L. J. Guibas, and D. T. Lee, "The power of geometric duality," in *Proc. 24th IEEE Annu. Symp. Found. Comput. Sci.*, Nov. 1983, pp. 217–225.
[87] B. M. Chazelle and J. Incerpi, "Unraveling the segment tree," Dep. Comput. Sci., Brown Univ., Providence, RI, Tech. Rep. CS-83-15, June 1983.
[88] ——, "Triangulating a polygon by divide-and-conquer," in *Proc. 21st Allerton Conf. Commun. Control Comput.*, Oct. 1983, pp. 447–456.
[89] B. M. Chazelle and D. T. Lee, "On a circle placement problem," in *Proc. Conf. Inform. Syst. Sci.*, Princeton, NJ, 1984.
[90] D. Cheriton and R. E. Tarjan, "Finding minimum spanning trees," *SIAM J. Comput.*, pp. 724–742, Dec. 1976.
[91] F. Chin and C. A. Wang, "Optimal algorithms for the intersection and the minimum distance problems between planar polygons," *IEEE Trans. Comput.*, vol. C-32, pp. 1203–1207, Dec. 1983.
[92] ——, "Minimum vertex distance between separable convex polygons," *Inform. Processing Lett.*, vol. 18, pp. 41–45, Jan. 1984.
[93] K. L. Clarkson, "Fast algorithms for the all nearest neighbors problems," in *Proc. 24th IEEE Annu. Symp. Found. Comput. Sci.*, Nov. 1983, pp. 226–232.
[94] R. Cole, "Slowing down sorting networks to obtain faster sorting algorithms," in *Proc. 25th IEEE Annu. Symp. Found. Comput. Sci.*, Oct. 1984, pp. 255–260.
[95] R. Cole and C. K. Yap, "Geometric retrieval problems," in *Proc. 24th IEEE Annu. Symp. Found. Comput. Sci.*, Nov. 1983, pp. 112–121.
[96] G. B. Dantzig, *Linear Programming and Extensions*. Princeton, NJ: Princeton Univ. Press, 1963.
[97] L. Devroye and G. T. Toussaint, "A note on linear expected time algorithms for finding convex hulls," *Computing*, vol. 26, pp. 361–366, 1981.
[98] A. K. Dewdney, "Complexity of nearest neighbor searching in three and higher dimensions," Univ. Western Ontario, Tech. Rep. 28, London, Ont., Canada, 1977.
[99] D. P. Dobkin, "A nonlinear lower bound on linear search tree programs for solving knapsack problems," *J. Comput. Syst. Sci.*, vol. 13, pp. 69–73, 1976.
[100] D. P. Dobkin, R. L. Drysdale III, and L. J. Guibas, "Finding smallest polygons," in *Advances in Computing Research, Vol. 1*, F. P. Preparata, Ed. JAI Press, 1983, pp. 181–214.
[101] D. P. Dobkin and H. Edelsbrunner, "Organizing points in two and three dimensions," IIG, Technische Univ. Graz, Austria, Rep. 130, Feb. 1984.
[102] ——, "Space searching for intersecting objects," manuscript.
[103] D. P. Dobkin and D. G. Kirkpatrick, "Fast detection of polyhedral intersection," *Theoret. Comput. Sci.*, vol. 27, pp. 241–253, 1983.
[104] ——, "A linear algorithm for determining the separation of convex polyhedra," *J. Algorith.*, to be published.
[105] ——, "Fast algorithms for preprocessed polyhedral intersection detection," in preparation.
[106] D. P. Dobkin and R. J. Lipton, "Multidimensional searching problems," *SIAM J. Comput.*, vol. 5, no. 2, pp. 181–186, June 1976.
[107] ——, "A lower bound of $1/2n^2$ on linear search programs for the knapsack problem," *J. Comput. Syst. Sci.*, vol. 16, pp. 413–417, 1978.
[108] ——, "On the complexity of computations under varying sets of primitives," *J. Comput. Syst. Sci.*, vol. 18, pp. 86–91, 1979.
[109] D. P. Dobkin, R. J. Lipton, and S. Reiss, "Linear programming is log-space hard for *P*," *Inform. Processing Lett.*, vol. 8, pp. 96–97, Feb. 1979.
[110] D. P. Dobkin and J. I. Munro, "Efficient use of the past," in *Proc. 21st IEEE Annu. Symp. Found. Comput. Sci.*, Oct. 1980, pp. 200–206.
[111] ——, "Optimal time minimal space selections algorithm," *J. ACM*, vol. 28, no. 3, pp. 454–461, July 1981.
[112] D. P. Dobkin and S. P. Reiss, "The complexity of linear programming," *Theoret. Comput. Sci.*, vol. 11, pp. 1–18, 1980.
[113] D. P. Dobkin and L. Snyder, "On a general method for maximizing and minimizing among certain geometric problems," in *Proc. 20th IEEE Annu. Symp. Found. Comput. Sci.*, 1979, pp. 9–17.
[114] Z. Drezner, "On a modified 1-center problem," *Manag. Sci.*, vol. 27, pp. 838–851, 1981.
[115] R. O. Duda and P. E. Hart, *Pattern Classification and Scene Analysis*. New York: Wiley, 1973.
[116] M. R. Dunlavey, "Query performance of a many-dimensional best-match algorithm," in *Proc. 19th Allerton Conf. Commun. Control Comput.*, 1981, pp. 389–396.
[117] M. E. Dyer, "A simplified $O(n \log n)$ algorithm for the intersection of 3-polyhedra," Dep. Math., Middlesbrough, UK, Tech. Rep. TPMR 80-5, 1980.
[118] ——, "Linear time algorithms for two- and three-variable linear programs," *SIAM J. Comput.*, vol. 13, no. 1, pp. 31–45, Feb. 1984.
[119] M. Edahiro, I. Kokubo, and T. Asano, "A new point-location algorithm

and its practical efficiency—Comparison with existing algorithms," Dep. Math. Eng. Instrumen. Phys., Univ. Tokyo, Tokyo, Japan, Res. Memo. RMI 83-04, Oct. 1983.
[120] W. Eddy, "A new convex hull algorithm for planar sets," *ACM Trans. Math. Software*, vol. 3, no. 4, pp. 398–403, Dec. 1977.
[121] H. Edelsbrunner, "Optimizing the dynamization of decomposable searching problems," IIG, Technische Univ. Graz, Austria, Rep. 35, Sept. 1979.
[122] ——, "A new approach to rectangle intersections, Part I," *Int. J. Comput. Math.*, vol. 13, pp. 209–219, 1983.
[123] ——, "A new approach to rectangle intersections, Part II," *Int. J. Comput. Math.*, vol. 13, pp. 221–229, 1983.
[124] ——, "Dynamic data structures for orthogonal intersection queries," IIG, Technische Univ. Graz, Austria, Rep. 59, Oct. 1980.
[125] ——, "A note on dynamic range searching," *Bull. EATCS*, vol. 15, pp. 34–40, 1981.
[126] ——, "Intersection problems in computational geometry," Ph.D. dissertation, IIG, Technische Univ. Graz, Austria, Rep. 93, 1982.
[127] ——, "On computing the extreme distances between two convex polygons," IIG, Technische Univ. Graz, Austria, Rep. 96, 1982.
[128] H. Edelsbrunner, L. J. Guibas, and J. Stolfi, "Optimal point location in a monotone subdivision," *SIAM J. Comput.*, to be published.
[129] H. Edelsbrunner and H. A. Maurer, "On the intersection of orthogonal objects," *Inform. Processing Lett.*, vol. 13, pp. 177–181, 1981.
[130] ——, "A space-optimal solution of general region location," *Theoret. Comput. Sci.*, vol. 16, pp. 329–336, 1981.
[131] H. Edelsbrunner, H. A. Maurer, and D. G. Kirkpatrick, "Polygonal intersection searching," *Inform. Processing Lett.*, vol. 14, pp. 74–79, 1982.
[132] H. Edelsbrunner, H. A. Maurer, F. P. Preparata, A. L. Rosenberg, E. Welzl, and D. Wood, "Stabbing line segments," *BIT*, vol. 22, pp. 274–281, 1982.
[133] H. Edelsbrunner, J. O'Rourke, and R. Seidel, "Constructing arrangements of lines and hyperplanes with applications," in *Proc. IEEE Annu. Symp. Found. Comput. Sci.*, Nov. 1983, pp. 83–91; also IIG, Technische Univ. Graz, Austria, Tech. Rep. F123, Sept. 1983.
[134] H. Edelsbrunner, T. Ottmann, J. van Leeuwen, and D. Wood, "Connected components of orthogonal geometric objects," Unit for Comput. Sci., McMaster Univ., Hamilton, Ont., Canada, Rep. 81-CS-04, 1981.
[135] H. Edelsbrunner and M. H. Overmars, "On the equivalence of some rectangle problems," *Inform. Processing Lett.*, vol. 14, no. 3, pp. 124–127, May 1982.
[136] ——, "Batched dynamic solutions to decomposable searching problems," *J. Algorith.*, to be published.
[137] H. Edelsbrunner, M. H. Overmars, and R. Seidel, "Some methods of computational geometry applied to computer graphics," IIG, Technische Univ. Graz, Austria, Tech. Rep. F117, June 1983.
[138] H. Edelsbrunner, M. H. Overmars, and D. Wood, "Graphics in flatland: A case study," in *Advances in Computing Research, Vol. 1*, F. P. Preparata, Ed. JAI Press, 1983, pp. 35–59.
[139] H. Edelsbrunner and J. van Leeuwen, "Multidimensional data structures and algorithms, A bibiliography," IIG, Technische Univ. Graz, Austria, Rep. 104, 1983.
[140] H. Edelsbrunner and E. Welzl, "Halfplanar range estimation," IIG, Technische Univ. Graz, Austria, Rep. 98, 1982.
[141] ——, "Halfplanar range search in linear space and $O(n^{0.695})$ query time," IIG, Technische Univ. Graz, Austria, Rep. 111, 1983.
[142] H. El Gindy, "An efficient algorithm for computing the weak visibility polygon from an edge in simple polygons," Jan. 1984.
[143] H. El Gindy and D. Avis, "A linear algorithm for computing the visibility polygon from a point," *J. Algorith.*, vol. 2, no. 2, pp. 186–197, June 1981.
[144] H. El Gindy, D. Avis, and G. T. Toussaint, "Applications of a two dimensional hidden-line algorithm to other geometric problems," *Computing*, vol. 31, pp. 191–202, 1983.
[145] H-Y. F. Feng and T. Pavlidis, "Decomposition of polygons into simpler components: Feature generation for syntactic pattern recognition," *IEEE Trans. Comput.*, vol. C-24, pp. 636–650, 1975.
[146] A. R. Forrest, "Computational geometry—Achievements and problems," in *Computer Aided Geometric Design*, R. E. Barnhill and R. F. Riesenfeld, Eds. New York: Academic, 1974, pp. 17–44.
[147] S. Fortune and J. Hopcroft, "A note on Rabin's nearest-neighbor algorithm," *Inform. Processing Lett.*, vol. 8, no. 1, pp. 20–23, Jan. 1979.
[148] R. L. Francis and J. A. White, *Facility Layout and Location*. Englewood Cliffs, NJ: Prentice-Hall, 1974.
[149] G. N. Frederickson and D. B. Johnson, "Finding kth paths and p-centers by generating and searching good data structures," *J. Algorith.*, vol. 4, pp. 61–80, 1983.
[150] M. L. Fredman and B. Weide, "On the complexity of computing the measure of $\cup[a_i, b_i]$," *Commun. ACM*, vol. 21, no. 7, pp. 540–544, 1978.
[151] M. L. Fredman, "A lower bound of the complexity of orthogonal range queries," *J. ACM.*, vol. 28, pp. 696–705, 1981.
[152] H. Freeman, "Computer processing of line-drawing images," *Comput. Surveys*, vol. 6, pp. 57–93, 1974.
[153] H. Freeman and P. P. Loutrel, "An algorithm for the two dimensional hidden line problem," *IEEE Trans. Electron. Comput.*, vol. EC-16, pp. 784–790, 1967.
[154] H. Freeman and R. Shapira, "Determining the minimum-area encasing rectangle for an arbitrary closed curve," *Commun. ACM*, vol. 18, pp. 409–413, July 1975.
[155] N. Friedman, "Some results on the effect of arithmetics on comparison problems," in *Proc. 13th IEEE Annu. Symp. Switching and Automata Theory*, 1972, pp. 139–143.
[156] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best match in logarithmic expected time," *ACM Trans. Math. Software*, vol. 3, no. 3, pp. 209–226, 1977.
[157] J. H. Friedman, F. Baskett, and J. Shustek, "An algorithm for finding nearest neighbors," *IEEE Trans. Comput.*, vol. C-24, pp. 1000–1006, 1975.
[158] H. N. Gabow, J. L. Bentley, and R. E. Tarjan, "Scaling and related techniques for geometry problems," *Proc. 16th ACM Annu. Symp. Theory Comput.*, Apr. 1984, pp. 135–143.
[159] R. Galimberti and U. Montanari, "An algorithm for hidden-line elimination," *Commun. ACM*, vol. 12, pp. 206–211, 1969.
[160] M. Garey, R. L. Graham, and D. S. Johnson, "Some NP-complete problems," in *Proc. 8th ACM Annu. Symp. Theory Comput.*, May 1976, pp. 10–22.
[161] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco, CA: Freeman, 1979.
[162] M. Garey, D. S. Johnson, F. P. Preparata, and R. E. Tarjan, "Triangulating a simple polygon," *Inform. Processing Lett.*, vol. 7, no. 4, pp. 175–180, 1978.
[163] P. D. Gilbert, "New results on planar triangulations," Coord. Sci. Lab., Univ. Illinois, Urbana, IL, Tech. Rep. ACT-15, July 1979.
[164] G. H. Gonnet, J. I. Munro, and D. Wood, "Direct dynamic structure for some line segment problems," *Comput. Vision, Graphics and Image Processing*, vol. 23, pp. 178–186, 1983.
[165] T. Gonzalez, "Algorithms on sets and related problems," Dep. Comput. Sci., Univ. Oklahoma, Tech. Rep., 1975.
[166] I. G. Gowda, D. G. Kirkpatrick, D. T. Lee, and A. Naamad, "Dynamic Voronoi diagrams," *IEEE Trans. Inform. Theory*, vol. IT-29, no. 5, pp. 724–731, Sept. 1983.
[167] R. L. Graham, "An efficient algorithm for determining the convex hull of a finite planar set," *Inform. Processing Lett.*, vol. 1, pp. 132–133, 1972.
[168] R. L. Graham and F. F. Yao, "Finding the convex hull of a simple polygon," Stanford Univ., Stanford, CA, Tech. Rep. STAN-CS-81-887, 1981; also *J. Algorith.*, vol. 4, no. 4, pp. 324–331, Dec. 1983.
[169] D. H. Greene, "The decomposition of polygons into convex parts," *Advances in Computing Research, Vol. 1*, F. P. Preparata, Ed. JAI Press, 1983, pp. 235–259.
[170] B. Grünbaum, *Convex Polytopes*. New York: Wiley Interscience, 1967.
[171] L. J. Guibas and J. Stolfi, "On computing all north-east nearest neighbors in the L_1-metric," *Inform. Processing Lett.*, vol. 17, pp. 219–223, Nov. 1983.
[172] R. H. Güting, "Optimal divide-and-conquer to compute measure and contour for a set of iso-rectangles," Lehrstuhl Informatik vi, Univ. Dortmund, Tech. Rep., 1982.
[173] ——, "Stabbing c-oriented polygons," *Inform. Processing Lett.*, vol. 16, pp. 35–40, Jan. 1983.
[174] R. H. Güting and D. Wood, "Finding rectangle intersections by divide-and-conquer," Univ. Waterloo, Waterloo, Ont., Canada, Tech. Rep. CS-83-03, 1983.
[175] J. A. Hartigan, *Clustering Algorithms*. New York: Wiley, 1975.
[176] S. Hertel, K. Mehlhorn, M. Mantyla, and J. Nievergelt, "Space sweep solves intersection of two convex polyhedra elegantly," *Acta Informatica*, to be published.
[177] F. K. Hwang, "An $O(n \log n)$ algorithm for rectilinear minimal spanning tree," *J. ACM*, vol. 26, pp. 177–182, 1979.
[178] H. Imai, "Finding connected components of an intersection graph of squares in the euclidean plane," *Inform. Processing Lett.*, vol. 15, no. 3, pp. 125–128, Oct. 1982.
[179] H. Imai and T. Asano, "Finding the connected components and a maximum clique of an intersection graph of rectangles in the plane,"

J. Algorith., vol. 4, no. 4, pp. 310–323, Dec. 1983.
[180] ——, "An efficient algorithm for finding a maximum matching of an intersection graph of horizontal and vertical line segments," in *Papers IECE Tech. Group Circuits and Systems,* CAS 83-143.
[181] ——, "Efficient algorithms for geometric graph search problems," Dep. Math. Eng. Instrumen. Phys., Univ. Tokyo, Tokyo, Japan, Res. Memo. RMI 83-05, Oct. 1983.
[182] ——, "Dynamic orthogonal segment intersection search," Dep. Math. Eng. Instrumen. Phys., Univ. Tokyo, Tokyo, Japan, Res. Memo. RMI 84-02, Feb. 1984.
[183] H. Imai, M. Iri, and K. Murota, "Voronoi diagram in the Laguerre geometry and its applications," *SIAM J. Comput.*, to be published.
[184] R. A. Jarvis, "On the identification of the convex hull of a finite set of points in the plane," *Inform. Processing Lett.*, vol. 2, pp. 18–21, 1973.
[185] G. H. Johansen and C. Gram, "A simple algorithm for building the 3-D convex hull," *BIT,* vol. 23, pp. 146–160, 1983.
[186] D. S. Johnson and F. P. Preparata, "The densest hemisphere problem," *Theoret. Comput. Sci.*, vol. 6, pp. 93–107, 1978.
[187] A. Jozwik, "A recursive method for the investigation of the linear separability of two sets," *Pattern Recog.*, vol. 11, no. 4, pp. 429–431, 1983.
[188] J. Kahn, M. Klawe, and D. Kleitman, "Traditional galleries require fewer watchman," *SIAM J. Algorith. Disc. Method,* vol. 4, no. 2, pp. 194–206, 1980.
[189] I. Kalantari and G. McDonald, "A data structure and an algorithm for nearest point problem," *IEEE Trans. Software Eng.*, vol. SE-9, no. 5, pp. 631–634, 1983.
[190] O. Kariv and S. L. Hakimi, "An algorithmic approach to network location problems, Part I: the centers," *SIAM J. Appl. Math.*, vol. 37, pp. 513–538, 1979.
[191] ——, "An algorithmic approach to network location problems, Part II: p-medians," *SIAM J. Appl. Math.*, vol. 37, pp. 539–560, 1979.
[192] J. Katajainen, "On the worst case of a minimal spanning tree algorithm for Euclidean space," *BIT*, vol. 23, pp. 2–8, 1983.
[193] J. M. Keil, "Decomposing polygons into simpler components," Ph.D. dissertation, Dep. Comput. Sci., Univ. Toronto, Toronto, Ont., Canada, 1983.
[194] J. M. Keil and J. R. Sack, "Minimum decompositions of polygonal objects," in *Computational Geometry,* G. T. Toussaint, Ed. Amsterdam, The Netherlands: North-Holland, to be published.
[195] L. G. Khachian, "A polynomial algorithm in linear programming," *Sov. Math. Dokl.*, vol. 20, pp. 191–194, 1979.
[196] D. G. Kirkpatrick, "Efficient computation of continuous skeletons," in *Proc. 20th IEEE Annu. Symp. Found. Comput. Sci.*, Oct. 1979, pp. 18–27.
[197] ——, "Optimal search in planar subdivisions," *SIAM J. Comput.*, vol. 12, no. 1, pp. 28–35, Feb. 1983.
[198] D. G. Kirkpatrick and R. Seidel, "The ultimate planar convex hull algorithm?" in *Proc. 20th Allerton Conf. Commun. Control Comput.*, 1982, pp. 35–42.
[199] ——, "The ultimate planar convex hull algorithm?" Dep. Comput. Sci., Cornell Univ., Ithaca, NY, Tech. Rep. 83-577, Oct. 1983.
[200] V. Klee, "On the complexity of d-dimensional Voronoi diagrams," *Archiv der Mathematik,* vol. 34, pp. 75–80, 1980.
[201] V. Klee and M. C. Laskowski, "Finding smallest triangles containing a given convex polygon," *J. Algorith.*, to be published.
[202] G. T. Klincsek, "Minimal triangulations of polygonal domains," *Ann. Discrete Math.*, vol. 9, pp. 121–123, 1980.
[203] D. E. Knuth, *The Art of Computer Programming: Sorting and Searching, Vol. 3.* Reading, MA: Addison-Wesley, 1973.
[204] ——, "Big omicron and big omega and big theta," *SIGACT News,* vol. 8.2, pp. 18–24, Apr.–June 1976.
[205] H. T. Kung, F. Luccio, and F. P. Preparata, "On finding the maxima of a set of vectors," *J. ACM,* vol. 22, pp. 469–476, 1975.
[206] R. C. Larson and V. O. K. Li, "Finding minimum rectilinear distance paths in the presence of barriers," *Networks,* vol. 11, no. 3, pp. 285–304, 1981.
[207] L. Lauther, "4-dimensional binary search trees as a means to speed up associative searches in design rule verification of integrated circuits," *J. Design Automat. Fault-Tolerant Computing,* vol. 2, pp. 241–247, 1978.
[208] C. L. Lawson, "C^1-compatible interpolation over a triangle," Jet Propulsion Lab., Tech. Memo. 33-770, May 1976.
[209] ——, "Integrals of a C^1-compatible triangular surface element," Jet Propulsion Lab., Tech. Memo. 33-808, Dec. 1976.
[210] ——, "Software for C^1 surface interpolation," Jet Propulsion Lab., pub. 77-30, Aug. 1977.
[211] D. T. Lee, "Proximity and reachability in the plane," Coord. Sci. Lab., Univ. Illinois, Urbana, IL, Tech. Rep. R-831, 1978.
[212] ——, "Two dimensional Voronoi diagram in the L_p-metric," *J. ACM.*, Oct. 1980, pp. 604–618.
[213] ——, "Farthest neighbor Voronoi diagrams and applications," Dep. Elec. Eng. Comput. Sci., Northwestern Univ., Evanston, IL, Tech. Rep. 80-11-FC-04, 1980.
[214] ——, "On finding the convex hull of a simple polygon," Dep. Elec. Eng. Comput. Sci., Northwestern Univ., Evanston, IL, Tech. Rep. 80-03-FC-01, 1980; see also *Int. J. Comput. Inform. Sci.*, vol. 12, no. 2, pp. 87–98, Apr. 1983.
[215] ——, "Shading of regions on vector display devices," *Comput. Graphics,* vol. 15, no. 3, pp. 37–44, Aug. 1981.
[216] ——, "On k-nearest neighbor Voronoi diagrams in the plane," *IEEE Trans. Comput.*, vol. C-31, pp. 478–487, June 1982.
[217] ——, "Medial axis transformation of a planar shape," *IEEE Trans. Pattern Anal. Machine Intell.*, vol. PAMI-4, no. 4, July 1982, pp. 363–369.
[218] ——, "Visibility of a simple polygon," *Comput. Vision, Graphics and Image Processing,* vol. 22, pp. 207–221, 1983.
[219] ——, "Maximum clique problem of rectangle graphs," in *Advances in Computing Research, Vol. 1*, F. P. Preparata, Ed. JAI Press, 1983, pp. 91–107.
[220] ——, "An optimal time and minimal space algorithm for rectangle intersection problems," *Int. J. Comput. Inform. Sci.*, to be published.
[221] D. T. Lee and I. M. Chen, "Display of visible edges of a set of convex polygons," in *Computational Geometry.* G. T. Toussaint, Ed. Amsterdam, The Netherlands: North-Holland, to be published.
[222] D. T. Lee and R. L. Drysdale III, "Generalized Voronoi diagram in the plane," *SIAM J. Comput.*, vol. 10, no. 1, pp. 73–87, Feb. 1981.
[223] D. T. Lee and A. Lin, "Computing visibility polygon from an edge," Dep. Elec. Eng. Comput. Sci., Northwestern Univ., Evanston, IL, Tech. Rep. 84-02-FC-01, Jan. 1984.
[224] D. T. Lee and N. Megiddo, "Routing around circles," manuscript, Oct. 1983.
[225] D. T. Lee and F. P. Preparata, "Location of a point in a planar subdivision and its applications," *SIAM J. Comput.*, vol. 6, no. 3, pp. 594–606, Sept. 1977.
[226] ——, "The all nearest neighbor problem for convex polygons," *Inform. Processing Lett.*, pp. 189–192, June 1978.
[227] ——, "An optimal algorithm for finding the kernel of a polygon," *J. ACM,* pp. 415–421, July 1979.
[228] ——, "An improved algorithm for the rectangle enclosure problem," *J. Algorith.*, vol. 3, no. 3, pp. 218–224, Sept. 1982.
[229] ——, "Euclidean shortest paths in the presence of rectilinear barriers," *Networks,* vol. 14, pp. 393–410, 1984.
[230] D. T. Lee and B. Schachter, "Two algorithms for constructing Delaunay triangulations," *Int. J. Comput. Inform. Sci.*, vol. 9, no. 3, pp. 219–242, June 1980.
[231] D. T. Lee and C. K. Wong, "Worst case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees," *Acta Informatica,* vol. 9, pp. 23–29, 1977.
[232] ——, "Voronoi diagrams in L_1-(L_∞-) metrics with 2-dimensional storage applications," *SIAM J. Comput.*, vol. 9, no. 1, pp. 200–211, Feb. 1980.
[233] ——, "Quintary trees: A file structure for multidimensional database systems," *ACM Trans. Database Syst.*, vol. 1, no. 3, pp. 339–353, Sept. 1980.
[234] ——, "Finding intersection of rectangles by range search," *J. Algorith.*, vol. 2, pp. 337–347, 1981.
[235] D. T. Lee and Y. F. Wu, "Efficient algorithms for Euclidean 1-line center and problems," in *Proc. ISOLDE III,* Boston, MA, 1984.
[236] A. Lingas, "The power of non-rectilinear holes," in *Proc. 9th Colloq. Automata, Lang. Programming,* Aarhus, Denmark, 1982.
[237] W. Lipski, Jr., "Finding a Manhattan path and related problems," *Networks,* vol. 13, pp. 399–409, 1983.
[238] W. Lipski, Jr., E. Lodi, F. Luccio, C. Mugnai, and L. Pagli, "On two dimensional data organization II," *Fundamenta Informaticae,* vol. 2, pp. 245–260, 1979.
[239] W. Lipski, Jr. and C. H. Papadimitriou, "A fast algorithm for testing for safety and detecting deadlocks in locked transaction systems," *J. Algorith.*, vol. 2, pp. 211–226, 1981.
[240] W. Lipski, Jr. and F. P. Preparata, "Finding the contour of a union of iso-oriented rectangles," *J. Algorith.*, vol. 1, pp. 235–246, 1980; see also *J. Algorith.*, vol. 3, p. 301, 1980.
[241] ——, "Segments, rectangles, contours," *J. Algorith.*, vol. 2, pp. 63–76, 1981.
[242] R. J. Lipton and R. E. Tarjan, "Applications of a planar separator theorem," in *Proc. 18th IEEE Annu. Symp. Found. Comput. Sci.*, 1977, pp. 162–170; see also *SIAM J. Comput.*, vol. 9, no. 3, pp. 615–627, Aug. 1980.

[243] T. Lozano-Perez and M. A. Wesley, "An algorithm for planning collision free paths among polyhedral obstacles," *Commun. ACM*, vol. 22, no. 10, pp. 560–570, Oct. 1979.
[244] G. S. Lueker and D. E. Willard, "A data structure for dynamic range queries," *Inform. Processing Lett.*, vol. 15, no. 5, pp. 209–213, Dec. 1982.
[245] U. Manber and M. Tompa, "Probabilistic, nondeterministic and alternating decision trees," Univ. Washington, Tech. Rep. 82-03-01, 1982.
[246] H. G. Mairson and J. Stolfi, "Reporting and counting line segment intersections," Dep. Comput. Sci., Stanford Univ., Stanford, CA, Extended Abstract, 1984.
[247] W. J. Masek, "Some NP-complete set covering problems," unpublished manuscript, Apr. 1979.
[248] H. A. Maurer and T. Ottmann, "Dynamic solutions of decomposable searching problems," in *Discrete Structures and Algorithms*, U. Pape, Ed. Hanser, 1979, pp. 17–24.
[249] D. McCallum and D. Avis, "A linear algorithm for finding the convex hull of a simple polygon," *Inform. Processing Lett.*, vol. 9 pp. 201–206, 1979.
[250] E. M. McCreight, "Efficient algorithms for enumerating intersecting intervals and rectangles," Xerox Palo Alto Res. Cen., Palo Alto, CA, Tech. Rep. PARC CSL-80-9, 1980.
[251] ——, "Priority search trees," Xerox Palo Alto Res. Cen., Palo Alto, CA, Tech. Rep. PARC CSL-81-5, 1981.
[252] M. McKenna and G. T. Toussaint, "Finding the minimum vertex distance between two disjoint convex polygons in linear time," School Comput. Sci., McGill Univ., Montreal, P. Q., Canada, Tech. Rep. SOCS-8306, Apr. 1983.
[253] D. H. McLain, "Two-dimensional interpolation from random data," *Comput. J.*, vol. 19, pp. 178–181, 1976.
[254] M. M. McQueen and G. T. Toussaint, "On the ultimate convex hull algorithm in practice," *Pattern Recog. Lett.*, to be published.
[255] N. Megiddo, "The weighted Euclidean 1-center problem," Dep. Statist., Tel Aviv Univ., Tel Aviv, Israel, 1981.
[256] ——, "On some planar location problems," Dep. Statist. Tel Aviv Univ., Tel Aviv, Israel, Oct. 1981.
[257] ——, "Linear time algorithm for linear programming in R^3 and related problems," *SIAM J. Comput.*, vol. 12, no. 4, pp. 759–776, Nov. 1983.
[258] ——, "Applying parallel computation algorithms in the design of serial algorithms," *J. ACM*, vol. 30, no. 4, pp. 852–865, Oct. 1983.
[259] ——, "Linear programming in linear time when the dimension is fixed," *J. ACM*, vol. 31, no. 1, pp. 114–127, Jan. 1984.
[260] N. Megiddo and K. J. Supowit, "On the complexity of some common geometric location problems," *SIAM J. Comput.*, vol. 13, no. 1, pp. 182–196, Feb. 1984.
[261] N. Megiddo, E. Zemel, and S. L. Hakimi, "The maximum coverage location problem," Cen. Math. Stud. Econ. Management Sci., Northwestern Univ., Evanston, IL, Discuss. Paper 490, Aug. 1981.
[262] K. Mehlhorn, "Lower bounds on the efficiency of static to dynamic transforms of data structures," *Math. Syst. Theory*, vol. 15, pp. 1–16, 1981.
[263] K. Mehlhorn and M. H. Overmars, "Optimal dynamization of decomposable searching problems," *Inform. Processing Lett.*, vol. 12, no. 2, pp. 93–98, Apr. 1981.
[264] M. L. Minsky and S. Papert, *Perceptron*. Cambridge, MA: M.I.T. Press, 1966.
[265] D. E. Muller and F. P. Preparata, "Finding the intersection of two convex polyhedra," *Theoret. Comput. Sci.*, vol. 7, pp. 217–236, 1978.
[266] A. Naamad, W. L. Hsu, and D. T. Lee, "On maximum empty rectangle problem," *Discrete Appl. Math.*, vol. 8, pp. 267–277, 1984.
[267] O. Nevalaninen, J. Ernvall, and J. Katajainen, "Finding minimal spanning trees in a Euclidean coordinate space," *BIT*, vol. 21, pp. 46–54, 1981.
[268] W. M. Newman and R. F. Sproull, *Principles of Interactive Computer Graphics*. New York: McGraw-Hill, 1979.
[269] J. Nievergelt and F. P. Preparata, "Plane-sweeping algorithms for intersecting geometric figures," *Commun. ACM*, vol. 25, no. 10, pp. 739–747, 1982.
[270] T. Ohtsuki, M. Sato, M. Tachibana, and S. Torii, "Minimum partitioning of rectilinear regions," *Trans. Inform. Processing Soc. Japan*, 1983.
[271] J. O'Rourke, "The complexity of computing minimum convex covers for polygons," in *Proc. 20th Allerton Conf. Commun. Control Comput.*, Oct. 1982, pp. 75–84.
[272] J. O'Rourke, A. Aggarwal, S. Maddila, and M. Baldwin, "An optimal algorithm for finding minimal enclosing triangles," Dep. Elec. Eng. Comput. Sci., Johns Hopkins Univ., Baltimore, MD, 1984, Tech. Rep. JHU/EECS-84/08, May 1984.
[273] J. O'Rourke, C-B. Chien, T. Olson, and D. Naddor, "A new linear algorithm for intersecting convex polygons," *Comput. Graph. Image Processing*, vol. 19, pp. 384–391, 1982.
[274] J. O'Rourke and K. J. Supowit, "Some NP-hard polygon decomposition problems," *IEEE Trans. Inform. Theory*, vol. IT-29, no. 2, pp. 181–190, Mar. 1983.
[275] T. Ottmann and D. Wood, "Dynamic sets of points," Dep. Comput. Sci., Univ. Waterloo, Waterloo, Ont., Canada, Tech. Rep. CS-82-56, Nov. 1982.
[276] M. H. Overmars, "Dynamization of order decomposable set problems," *J. Algorith.*, vol. 2, pp. 245–260, Sept. 1981; see also *J. Algorith.*, vol. 4, p. 301, Sept. 1983.
[277] ——, "Range searching in a set of line segments," Univ. Utrecht, Utrecht, The Netherlands, Tech. Rep. RUU-CS-83-6, Feb. 1983.
[278] ——, "The locus approach," Univ. Utrecht, Utrecht, The Netherlands, Tech. Rep. RUU-CS-83-12, 1983.
[279] M. H. Overmars and J. van Leeuwen, "Two general methods for dynamizing decomposable searching problems," *Computing*, vol. 26, pp. 155–166, 1981.
[280] ——, "Some principles for dynamizing decomposable searching problems," *Inform. Processing Lett.*, vol. 12, pp. 49–54, 1981.
[281] ——, "Maintenance of configurations in the plane," *J. Comput. Syst. Sci.*, vol. 23, pp. 166–204, 1981.
[282] ——, "Dynamization of decomposable searching problems yielding good worst-case bounds," *Lecture Notes in Computer Science 104*. New York: Springer Verlag, 1981, pp. 224–233.
[283] ——, "Worst-case optimal insertion and deletion methods for decomposable searching problems," *Inform. Processing Lett.*, vol. 12, pp. 168–173, 1981.
[284] C. H. Papadimitriou, "The Euclidean traveling salesman problem is NP-complete," *Theoret. Comput. Sci.*, vol. 4, pp. 237–244, 1977.
[285] ——, "Worst-case and probablistic analysis of a geometric location problem," *SIAM J. Comput.*, vol. 10, pp. 542–557, 1981.
[286] T. Pavlidis, *Algorithms for Graphics and Image Processing*. Berlin: Springer Verlag, 1982.
[287] M. J. Post, "A minimum spanning ellipse algorithm," in *Proc. 22nd IEEE Annu. Symp. Found. Comput. Sci.*, Oct. 1981, pp. 115–122.
[288] ——, "Minimum spanning ellipsoids," in *Proc. 16th Symp. Theory Comput.*, Apr. 1984, pp. 108–116.
[289] M. J. D. Powell and M. A. Sabin, "Pairwise quadratic approximation on triangles," *ACM Trans. Math. Software*, vol. 3, no. 4, pp. 316–325, 1977.
[290] F. P. Preparata, "The medial axis of a simple polygon," in *Proc. 6th Symp. Math. Found. Comput. Sci., Lecture Notes in Computer Science 53*. New York: Springer Verlag, 1977, pp. 443–450.
[291] ——, "An optimal real time algorithm for planar convex hulls," *Commun. ACM*, vol. 22, pp. 402–405, 1979.
[292] ——, "A new approach to planar point location," *SIAM J. Comput.*, vol. 10, no. 3, pp. 473–482, Aug. 1981.
[293] F. P. Preparata and S. J. Hong, "Convex hulls of finite sets of points in two and three dimensions," *Commun. ACM*, vol. 20, no. 2, pp. 87–93, Feb. 1977.
[294] F. P. Preparata and D. E. Muller, "Finding the intersection of a set of N half-spaces in time $O(N \log N)$," *Theoret. Comput. Sci.*, vol. 8, pp. 45–55, 1979.
[295] F. P. Preparata and M. I. Shamos, *Computational Geometry*. New York: Springer-Verlag, 1985, to be published.
[296] M. O. Rabin, "Proving simultaneous positivity of linear forms," *J. Comput. Syst. Sci.*, vol. 6, pp. 639–650, 1972.
[297] V. Raghavan and C. Y. Yu, "A note on a multidimensional searching problem," *Inform. Processing Lett.*, vol. 6, no. 4, pp. 133–135, Aug. 1977.
[298] E. M. Reingold, "On the optimality of some set algorithms," *J. ACM*, vol. 19, no. 4, pp. 649–659, Oct. 1972.
[299] R. Riesenfeld, "Applications of b-spline approximation to geometric problems of computer-aided design," Dep. Comput. Sci., Univ. Utah, Salt Lake City, UT, Tech. Rep. UTEC-CSc-73-126, 1973.
[300] C. A. Roger, *Packing and Covering*. Cambridge, England: Cambridge University Press, 1964.
[301] A. Rosenfeld, *Picture Processing by Computers*. New York: Academic, 1969.
[302] J. R. Sack, "An $O(n \log n)$ algorithm for decomposing simple rectilinear polygons into convex quadrilaterals," in *Proc. 20th Allerton*

Conf. Commun. Control Comput., 1982, pp. 64–74.
[303] J. R. Sack and G. T. Toussaint, "A linear time algorithm for decomposing rectilinear star-shaped polygons into convex quadrilaterals," in *Proc. 19th Allerton Conf. Commun. Control Comput.*, 1981, pp. 21–30.
[304] J. B. Saxe and J. L. Bentley, "Transforming static data structures into dynamic structures," in *Proc. 20th IEEE Annu. Symp. Found. Comput. Sci.*, Oct. 1979, pp. 148–168.
[305] M. Schlag, F. Luccio, P. Maestrini, D. T. Lee, and C. K. Wong, "A visibility problem in VLSI layout compaction," *Advances in Computing Research, Vol. 2*, F. P. Preparata, Ed. JAI Press, to be published.
[306] A. Schonhage, M. Paterson, and N. Pippenger, "Finding the median," *J. Comput. Syst. Sci.*, vol. 13, pp. 184–199, 1976.
[307] A. A. Schoone and J. van Leeuwen, "Triangulating a star-shaped polygon," Univ. Utrecht, Utrecht, The Netherlands, Tech. Rep. RUV-CS-80-3, Apr. 1980.
[308] J. T. Schwartz, "Finding the minimum distance between two convex polygons," *Inform. Processing Lett.*, vol. 13, no. 4, pp. 168–170, 1981.
[309] J. T. Schwartz and M. Sharir, "On the piano movers problem," Dep. Comput. Sci, Courant Inst. Math. Sci., New York Univ., New York, NY, Tech. Rep. 41, Feb. 1982.
[310] R. Seidel, "A convex hull algorithm optimal for points in even dimensions," M.S. thesis, Dep. Comput. Sci., Univ. British Columbia, Vancouver, B.C., Canada, Tech. Rep. 81-14, 1981.
[311] ——, "The complexity of Voronoi diagrams in higher dimensions," in *Proc. 20th Allerton Conf. Commun. Control and Comput.*, 1982, pp. 94–95; see also Technische Univ. Graz, Graz, Austria, Tech. Rep. F94, July 1982.
[312] ——, "A method for lower bounds for certain geometric problems," Dep. Comput. Sci., Cornell Univ., Ithaca, NY, Tech. Rep. 84-592, Feb. 1984.
[313] M. I. Shamos, "Geometric complexity," in *Proc. 7th ACM Annu. Symp. Theory Comput.*, May 1975, pp. 224–233.
[314] ——, "Geometry and statistics: Problems at the interface," in *Algorithms and Complexity*, J. F. Traub, Ed. New York: Academic, 1976, pp. 251–280.
[315] ——, "Computational geometry," Ph.D. dissertation, Dep. Comput. Sci., Yale Univ., New Haven, CT, 1978.
[316] M. I. Shamos and D. Hoey, "Closest-point problems," in *Proc. 16th IEEE Annu. Symp. Found. Comput. Sci.*, Oct. 1975, pp. 151–162.
[317] ——, "Geometric intersection problems," in *Proc. 17th IEEE Annu. Symp. Found. Comput. Sci.*, Oct. 1976, pp. 208–215.
[318] M. Sharir and A. Schorr, "On shortest paths in polyhedral spaces," in *Proc. 16th ACM Annu. Symp. Theory Comput.*, Apr. 1984, pp. 144–153.
[319] B. W. Silverman and D. M. Titterington, "Minimum covering ellipses," *SIAM J. Sci. Statist. Comput.*, vol. 1, no. 4, pp. 401–409, Dec. 1980.
[320] H. W. Six and D. Wood, "The rectangle intersection problem revisited," *BIT*, vol. 20, pp. 426–433, 1980.
[321] ——, "Counting and reporting intersections of d-ranges," *IEEE Trans. Comput.*, vol. C-31, pp. 181–187, 1982.
[322] W. E. Snyder and D. A. Tang, "Finding the extrema of a region," *IEEE Trans. Pattern Recog. Mach. Intell.*, vol. PAMI-2, pp. 266–269, May 1980; see also *IEEE Trans. Pattern Recog. Mach. Intell.*, vol. PAMI-4, p. 309, May 1982.
[323] E. Soisalon-Soininen and D. Wood, "An optimal algorithm to compute the closure of a set of iso-rectangles," *J. Algorith.*, vol. 5, no. 2, pp. 199–214, June 1984.
[324] J. M. Steele and A. C. Yao, "Lower bounds for algebraic decision trees," *J. Algorith.*, vol. 3, pp. 1–8, 1982.
[325] K. J. Supowit, "Grid heuristics for some geometricv covering problems," in *Advances in Computing Research, Vol. 1*, F. P. Preparata, Ed. JAI Press, 1983, pp. 215–233.
[326] I. Sutherland, R. Sproull, and R. Schumacker, "A characterization of ten hidden-surface algorithms," *Comput. Surveys*, vol. 6, no. 1, pp. 1–55, 1974.
[327] M. Tompa, "An optimal solution to a wire-routing problem," *J. Comput. Syst. Sci.*, vol. 23, pp. 127–150, 1981.
[328] G. T. Toussaint, "Pattern recognition and geometrical complexity," in *Proc. 5th Int. Conf. Pattern Recog.*, Dec. 1980, pp. 1324–1347.
[329] ——, "Computational geometric problems in pattern recognition," in *Pattern Recognition Theory and Applications*, J. Kitter, Ed. Oxford, England: NATO ASI, Apr. 1981.
[330] ——, "Solving geometric problems with the rotating calipers," in *Proc. IEEE MELECON '83*, Athens, Greece, May 1983.
[331] ——, "An optimal algorithm for computing the minimum vertex distance between two crossing convex polygons," in *Proc. 21st Allerton Conf. Commun. Control and Comput.*, Oct. 1983, pp. 457–458.
[332] ——, "Computing largest empty circle with location constraints," *Int. J. Comput. Inform. Sci.*, vol. 12, no. 5, pp. 347–358, Oct. 1983.
[333] ——, "A historical note on convex hull finding algorithms," *Pattern Recog. Lett.*, to be published.
[334] G. T. Toussaint and D. Avis, "On a convex hull algorithm for polygons and its applications to triangulation problems," *Pattern Recog.*, vol. 15, pp. 23–29, 1982.
[335] V. Vaishnavi, "Computing point enclosures," *IEEE Trans. Comput.*, vol. C-31, pp. 22–29, Jan. 1982.
[336] V. Vaishnavi and D. Wood, "Data structures for the rectangle containment and enclosure problems," *Comput. Graph. Image Processing*, vol. 13, pp. 372–384, 1980.
[337] ——, "Rectilinear line segment intersection, layered segment trees, and dynamization," *J. Algorith.*, vol. 3, pp. 160–176, 1982.
[338] P. van Emde Boas, "On the $\Omega(n \log n)$ lower bound for convex hull and maximal vector determination," *Inform. Processing Lett.*, vol. 10, pp. 132–136, 1980.
[339] J. van Leeuwen and H. A. Maurer, "Dynamic systems of static data structures," Technische Univ. Graz, Graz, Austria, Rep. 42, 1980.
[340] J. van Leeuwen and M. H. Overmars, "The art of dynamizing," in *Mathematical Foundations of Computer Science, Lecture Notes in Computer Science, Vol. 118*, J. Gruska and M. Chytil, Eds. Heidelberg, Germany: Springer-Verlag, 1981, pp. 121–131.
[341] J. van Leeuwen and D. Wood, "Dynamization of decomposable searching problems," *Inform. Processing Lett.*, vol. 10, pp. 51–56, 1980.
[342] ——, "The measure problem for rectangular ranges in d-space," *J. Algorith.*, vol. 2, pp. 282–300, 1981.
[343] D. F. Watson, "Computing the n-dimensional Delaunay tessellation with applications to Voronoi Polytopes," *Comput. J.*, vol. 24, no. 2, pp. 167–172, 1981.
[344] D. E. Willard, "Predicate-oriented database search algorithms," Ph.D. dissertation, Harvard Univ., Cambridge, MA, 1978.
[345] ——, "Polygon retrieval," *SIAM J. Comput.*, vol. 11, pp. 149–165, 1982.
[346] ——, "New data structures for orthogonal queries," *SIAM J. Comput.*, to be published.
[347] D. E. Willard and G. S. Lueker, "Adding range restriction capability to dynamic data structures," *J. ACM*, to be published.
[348] C. C. Yang and D. T. Lee, "A note on all nearest neighbor problem for convex polygons," *Inform. Processing Lett.*, vol. 8, pp. 193–194, Apr. 1979.
[349] A. C. Yao, "On the complexity of comparison problems using linear functions," in *Proc. 16th Annu. Symp. Theory Comput.*, 1975, pp. 85–89.
[350] ——, "A lower bound to finding convex hulls," *J. ACM*, vol. 28, pp. 780–787, 1981.
[351] ——, "On constructing minimum spanning tree in k-dimensional space and related problems," *SIAM J. Comput.*, vol. 11, no. 4, pp. 721–736, 1982.
[352] A. C. Yao and R. L. Rivest, "On the polyhedral decision problem," *SIAM J. Comput.*, vol. 9, pp. 343–3471, 1980.
[353] F. F. Yao, "A 3-space partition and its applications," in *Proc. 15th ACM Annu. Symp. Theory Comput.*, Apr. 1983, pp. 258–263.
[354] G. Yuval, "Finding nearest neighbours," *Inform. Processing Lett.*, vol. 5, no. 3, pp. 63–65, 1976.

D. T. Lee (S'76–M'78–SM'84), for a photograph and biography, see p. 6 of the January 1984 issue of this TRANSACTIONS.

Franco P. Preparata (M'63–SM'71–F'78), for a photograph and biography, see p. 437 of the May 1984 issue of this TRANSACTIONS.