LABORATORY FOR
COMPUTER SCIENCE

MASSACHUSETTS
INSTITUTE OF
TECHNOLOGY

MIT/LCS/TM-388

THREE METHODS FOR RANGE QUERIES IN COMPUTATIONAL GEOMETRY

Shlomo Kipnis

March 1989

545 TECHNOLOGY SQUARE, CAMBRIDGE, MASSACHUSETTS 02139

Three Methods for Range Queries in Computational Geometry

Shlomo Kipnis

Laboratory for Computer Science
Massachusetts Institute of Technology
Cambridge, Massachusetts 02139

March 27, 1989

Abstract

This paper surveys a variety of recent results addressing the problem of range queries in computational geometry. The major contribution of this paper is in identifying three general methods for range queries in computational geometry and in classifying many of the recent results into one or more of these approaches. The three methods discussed in this paper are random sampling, search-tree tables, and space-partition trees. This survey assumes some familiarity with basic computational geometry concepts and techniques.

1 Introduction

A widely studied area of algorithmic design concerns the construction of efficient data structures for maintaining a database of records that supports various types of queries. A variety of sophisticated data structures and algorithmic techniques for database queries have been recently developed.

A rather general class of database queries is the class of range queries. A range query is a request to identify the records in a database, with keys falling in a certain range. When each record has one key, there are various schemes for efficiently processing such range queries (balanced binary trees are a typical example). The problem becomes more complicated, however, when records have $d > 1$ keys and range queries involve general ranges in the d-dimensional space of keys. Multidimensional range queries, and in particular orthogonal range queries, where ranges are generalized rectangles in d dimensions, received a considerable amount of attention (see [2, 3, 4, 5, 19, 22, 23, 24, 28, 29]).

This research was supported in part by the Defense Advance Research Projects Agency under Contract N00014-87-K-0825.

Recently, the problem of multidimensional range queries was also intensively studied in the domain of computational geometry. The problem of range queries in computational geometry can be informally described as follows: given a collection of n geometrical objects in the d-dimensional Euclidean space, store them in a data structure, such that when given a range (subset) of the d-dimensional Euclidean space, the objects falling in that range can be quickly identified. The n geometrical objects are considered to be the records of a database, and the subset of the space is the query range. A formal definition of range queries is given in Section 3.

Throughout this paper we consider a prototypical example, the half-space range query problem. There are two important variants of this problem, half-space counting and half-space reporting. The half-space counting problem can be stated as follows: given n points in the d-dimensional Euclidean space E^d, organize them in a data structure, such that the *number* of points in a query half-space can be quickly determined. The half-space reporting problem is similar, except that the points in the query half-space need to be actually reported. Other common variants involve checking whether there exists a point that falls in the query half-space, checking whether all the points fall in the query half-space, and finding a point that minimizes a given function with respect to the query half-space. In Section 3, we define these types of queries more precisely.

Ranges in computational geometry are usually defined by analytic curves, and can thus be quite general. However, efficient algorithms and data structures for processing the corresponding range queries are known only for some specific types of ranges. These types of ranges include half-spaces, hyperrectangular (orthogonal) ranges, hyperspherical ranges, simplicial ranges, and polyhedral ranges with a bounded number of faces. Although it seems at first that these different types of ranges have little in common, it is possible, for example, to transform some types of range queries to others. These transformations usually involve mapping objects in the d-dimensional Euclidean space to points in some higher dimensional Euclidean space. Some examples of these transformations are given in [17, 35, 36]. Dobkin & Edelsbrunner [17], for instance, showed that a variety of queries in E^2 involving points intersecting triangles, segments intersecting segments, and polygons intersecting polygons, as well as related problems in E^3, can be reduced to half-space range queries. Yao [35] showed that polyhedral range queries can be solved by a sequence of half-space range queries, and that circular region queries in E^2 can be reduced to half-space range queries in E^3. Yao & Yao [36] gave a general scheme of reducing a variety of geometrical queries in E^d to range queries in higher dimensional Euclidean spaces.

Efficient algorithms for range queries are of primary interest in the field of computational geometry. Besides being important in their own right, many other problems in computational geometry either use range queries as subroutines or use generalized versions of range queries. Examples include constructing order-k Voronoi diagrams in the plane (Lee [27]), finding a Minimal Spanning Tree in E^d (Yao [33]), various geometric intersection problems in E^2 and E^3 (Dobkin & Edelsbrunner [17]), polytope-polytope intersection in E^d and the nearest neighbor problem (Yao & Yao [36]). Some types of range queries can be themselves reduced to other problems in computational geometry. For example, the problem of half-space queries can be solved by reducing it to the point location problem and building a data structure for point location searches. This reduction forms the basis

of some algorithms for the half-space query problem discussed in Section 5.

The term "efficient algorithms for range queries" needs a little clarification. The natural complexity parameter of an algorithm for range queries is n, the number of objects in the database. We generally assume that the database of objects will be queried many times, such that it is worth to initially invest the time for preprocessing the objects and to allocate the storage for the data structure. A solution to a range query problem is usually evaluated by its *query time*, its *storage*, and its *preprocessing time*. The main interest is in sublinear query time, since for most range queries we can determine in constant time whether an object falls in a specified range, which leads to a straightforward linear query time algorithm (exhaustive search). When discussing the query time, it is important to distinguish between range counting queries and range reporting queries. In the former case, for example, there are algorithms with sublinear query time, which is independent of the answer size. In the latter case, however, the query time contains two components, the search time and the number of points reported. The query time for range reporting problems, is thus of the form $O(f(n) + k)$, where $f(n)$, the search time, may be sublinear and independent of the specific query, and k, the report time, is dependent on the specific query. The storage requirement is usually polynomial in n. There is a particular class of algorithms for range queries, which we discuss in Section 6, that achieves the optimal, linear storage. The preprocessing time is usually also polynomial in n, but is considered as less important measure of an algorithm, as it represents only a one-time effort. Some relationships between the query time and the storage are presented in section 7.

In this paper, we study the problem of range queries in computational geometry. In particular, we focus on half-space and simplicial range queries, which constitute a rather large class of range queries. We describe, compare, and analyze three general methods for processing certain classes of range queries in computational geometry. The three methods are random sampling, search-tree tables, and space-partition trees. In the course of discussing these three techniques, we survey and classify a variety of results for range query problems in computational geometry. The different methods may be sometimes combined and can be related in some cases. All the results presented use the *Real Random Access Machine* model of computation, which is the most commonly used model in computational geometry. This model is an extension of the usual RAM model with unlimited precision real arithmetic operations and storage in constant time.

The remainder of this paper in organized as follows. Section 2 introduces commonly used geometric notations and some relevant problems and techniques in computational geometry. In Section 3, we define the notion of a range query in abstract range spaces and discuss some properties of range spaces and range queries. Section 4 presents the method of random sampling for range queries. In Section 5, we describe the method of search-tree tables for range queries. Section 6 discusses the method of space-partition trees for range queries. In Section 7, we present some lower bounds on the query time and storage requirements of algorithms for range queries. Finally, Section 8 summarizes and compares the different methods.

2 Geometric Fundamentals

In this section we introduce commonly used geometric notations, define some terms specific to our discussion, and describe some relevant techniques in computational geometry. The reader may wish to only skim this section at first, in order to become familiar with the notations used in this paper, and later refer to specific terms as necessary.

Spaces, Points, and Point Sets. We use E^d to denote the real d-dimensional Euclidean space with the standard metric L_2. Points of E^d are d-dimensional vectors over the reals. For a point set $A \subset E^d$, we use $affA$ to denote the affine closure of A, and we use $dimA$ to denote the affine dimension of A. A *k-flat* is a point set $A \subset E^d$, with $A = affA$ and $k = dimA$.

Hyperplanes and Half-Spaces. A *hyperplane* h in E^d is a *$(d-1)$-flat*, defined by the equation $\sum_{i=1}^{d} a_i x_i + b = 0$ for some values of $a_1, a_2, \ldots, a_d$. The hyperplane h separates E^d into a right open half-space h^+, defined by $\sum_{i=1}^{d} a_i x_i + b > 0$ and a left open half-space h^-, defined by $\sum_{i=1}^{d} a_i x_i + b < 0$. We use h^* to refer to either one of these two open half-spaces, and $\bar{h}^*$ to refer to either one of the closed half-spaces. We use the notation H_d to denote the collection of all hyperplanes in E^d, the notation H_d^* to denote all the open half-spaces in E^d, and the notation $\bar{H}_d^*$ to denote all the closed half-spaces in E^d.

General Position Assumption. In computational geometry it is usually assumed that the points are in general position. This means that all points are distinct, no three points are collinear, no four points are coplanar, and in general, no $k+1$ points lie in the same *$(k-1)$-flat*, for $k \geq 1$. This is not a restrictive assumption, as small random perturbations assure the above conditions with probability 1. Similarly, hyperplanes are assumed to be in general position, that is, the intersection of any $r \leq d$ hyperplanes in E^d is a *k-flat* of dimension $k = d - r$.

Convex Set, Convex Hull, and k-Set. A domain $A \subset E^d$ is *convex* if for any two points $x, y \in A$, the line segment between x and y is entirely contained in A. The *convex hull* of a point set A is the boundary of the minimal convex domain in E^d containing A. A *k-set* of a point set A is any subset $B \subset A$ of cardinality $|B| = k$, such that there is a hyperplane h, separating B from $A - B$, that is, $B = A \cap h^*$.

Polyhedral Sets and Polytopes. A *polyhedral set* is the intersection of a finite number of closed half-spaces, and a *polytope* is a bounded polyhedral set. A *face* of a polyhedral set P is the intersection of P with one or more of the hyperplanes defining it. Vertices, edges, and facets of a polyhedral set of dimension d are faces of affine dimensions $0, 1$, and $d-1$, respectively.

Complex. A *complex* is a collection of polyhedral sets, such that every face of a polyhedral set in the complex is also in the complex, and the intersection of two polyhedral sets in the complex is a face shared by each of them.

Simplex and Triangulation. A *k-simplex* in E^d (for $k \leq d$) is the convex hull of $k+1$ affinely independent points. Any r-dimensional face of a k-simplex is an r-simplex. When the dimension of the k-simplex is not important, we refer to it as just a *simplex*. A

triangulation $\Delta(C)$ of a complex C is a refinement of C into simplicial regions, such that each polyhedral set of dimension k in $\Delta(C)$ is a k-simplex.

Arrangement of Hyperplanes. For a collection H of n hyperplanes in E^d, the arrangement of H, $\mathcal{A}(H)$, is the complex generated by the cells defined by the hyperplanes of H. For representing arrangements, we usually keep lists of polyhedral regions and neighborhood relations. For $d \geq 0$ and $n \geq 0$, we define $\Phi_d(n)$ to be $\sum_{k=0}^{d} \binom{n}{k}$ if $d < n$ and 2^n otherwise. For a collection H of n hyperplanes in general position in E^d, the arrangement $\mathcal{A}(H)$ has $\Phi_d(n)$ cells. Furthermore, if h is any other hyperplane in E^d, then the number of cells in the arrangement $\mathcal{A}(H)$ intersected by h is at most $\Phi_{d-1}(n)$.

Point Hyperplane Duality. Points in E^d can be transformed to hyperplanes in H_d and vice-versa. The most common of these transformations, *Hough transformation*, maps a point $(a_1, a_2, \ldots, a_d)$ in E^d to the hyperplane $\sum_{k=1}^{d} a_i x_i + 1 = 0$ in H_d, and conversely. The duality relationship between points and hyperplanes allows a problem in the points space to be solved by first transforming the points into the hyperplanes domain, and then solving a related problem in the dual space. Similarly, in the other direction, problems on hyperplanes can be transformed into related problems on points.

Point Location Problem. Given an underlying partition of the space into regions (e.g. Voronoi diagrams, grids, or arrangements of hyperplanes), the *point location problem* is to identify the region in which a query point falls.

3 Range Spaces and Range Queries

In this section we formally define the notions of range space and range query and present some properties of range spaces. We relate these notions and properties to range queries in computational geometry.

We first define the notion of a range space. The following definition is from Haussler & Welzl [25] and is based on the pioneering work by Vapnik & Chervonenkis [30].

Definition 1 A *range space* $\mathcal{T}$ is a pair $(\mathcal{V}, \mathcal{R})$, where $\mathcal{V}$ is a set and $\mathcal{R}$ is a set of subsets of $\mathcal{V}$. Members of $\mathcal{V}$ are called *elements* or *points* of $\mathcal{T}$ and members of $\mathcal{R}$ are called *ranges* of $\mathcal{T}$. The range space $\mathcal{T}$ is finite if $\mathcal{V}$ is finite.

In the domain of computational geometry, we usually use the range space $\mathcal{T} = (E^d, \mathcal{R})$, where elements of $\mathcal{T}$ are points in the d-dimensional Euclidean space, and $\mathcal{R}$ is some family of regions in E^d.

We next present the definition of the Vapnik-Chervonenkis dimension of an abstract range space. Recently, it became evident that the notion of the V-C dimension of a range space is an important and useful parameter of the space.

Definition 2 Let $\mathcal{T} = (\mathcal{V}, \mathcal{R})$ be a range space and let $X \subset \mathcal{V}$ be a finite set of elements of $\mathcal{T}$. We denote by $\Pi_{\mathcal{R}}(X)$ the set of all subsets of X that can be obtained by intersecting

X with a range of $\mathcal{T}$, that is , $\Pi_{\mathcal{R}}(X) = \{X \cap Y : Y \in \mathcal{R}\}$. If $\Pi_{\mathcal{R}}(X) = 2^X$, then we say that X is *shattered* by $\mathcal{R}$. The *Vapnik-Chervonenkis dimension* of $\mathcal{T}$ (or simply the *V-C dimension*) is the largest integer d such that there exist a subset X of $\mathcal{V}$ of cardinality d that is shattered by $\mathcal{R}$. If no such maximal d exists, we say the dimension of $\mathcal{T}$ is infinite.

Applying Definition 2 to range spaces in computational geometry gives some interesting results. For example, the V-C dimension of the range space $\mathcal{T}_1 = (E^d, H_d^*)$, consisting of all open half-spaces as ranges, is $d + 1$. As another example, the V-C dimension of $\mathcal{T}_2 = (E^2, C_3)$, consisting of all triangular regions in the Euclidean plane, is 7. The latter example can be verified by noticing that any subset of 7 points, equally spaced on the circumference of a circle in E^2, can be separated from the other points by drawing an appropriate triangle that surrounds them. For 8 points in E^2, it is no longer possible to separate any subset of them with a triangle.

We now introduce a formal definition for the notion of a range query in range spaces. This definition is a combination of ideas from Fredman [24] and Haussler & Welzl [25].

Definition 3 Let $\mathcal{T} = (\mathcal{V}, \mathcal{R})$ be a range space, $(S, +)$ be a commutative semigroup, and f be a function $f : \mathcal{V} \times \mathcal{R} \rightarrow S$. A *database* is a finite set $X \subset \mathcal{V}$. A *range query* Q on a database $X \subset \mathcal{V}$ and a range $Y \in \mathcal{R}$ is defined as $Q(X, Y) = \sum_{x \in X} f(x, Y)$, where the sum is the semigroup sum.

The interpretation of Definition 3 in the domain of computational geometry is as follows. A database X is a finite set of points from E^d. A range query is defined on a database X and a range (subset) of E^d. The general outcome of a range query is considered as a value from some commutative semigroup $(S, +)$. The function f maps any pair (x, Y) of a point and a range to some element in S. The choice of the function f and the semigroup $(S, +)$ depends on the intended purpose of the data structure. Several examples of selecting f and $(S, +)$ for commonly used range queries are listed below.

- For counting the number of elements of X in the range Y, we define $f(x, Y)$ to be 1 if x is in range Y and 0 otherwise. We choose $(S, +)$ to be $(\mathbf{Z}, +)$, the integers with the usual addition operation
- For reporting the elements of X in the range Y, we define $f(x, Y)$ to be $\{x\}$ if x is in range Y and $\emptyset$ otherwise. We choose $(S, +)$ to be $(2^X, \cup)$, the collection of all subsets of X with the set union operation.
- For checking whether there exists an element of X in the range Y, we define $f(x, Y)$ to be 1 if x is in range Y and 0 otherwise. We choose $(S, +)$ to be $(\{0, 1\}, \vee)$.
- For checking whether all the elements of X are in the range Y, we define $f(x, Y)$ to be 1 if x is in range Y and 0 otherwise. We choose $(S, +)$ to be $(\{0, 1\}, \wedge)$.
- For computing the minimal value of some real function $g(x, Y)$ over all elements of X and the range Y, we define $f(x, Y) = g(x, Y)$. We choose $(S, +)$ to be $(\mathbf{R}, \min)$, the reals with the minimum operation.

The definition of range queries is rather general; all the types of range queries mentioned in the introduction are special cases of Definition 3. For all these range queries we use the range space $\mathcal{T} = (E^d, \mathcal{R})$ mentioned above. The ranges $\mathcal{R}$ of $\mathcal{T}$ can be chosen as half-spaces defined by hyperplanes in E^d, hyperrectangular (orthogonal) regions defined by $2d$ real numbers, hyperspherical regions defined by their center point in E^d and their radius, simplicial regions defined by $d+1$ points in E^d, and polyhedral regions defined by a finite collection of hyperplanes in E^d.

In the rest of the paper, we restrict ourselves to half-space and simplex queries, and comment on how some of the results can be extended to other ranges. As was mentioned in the introduction, some range queries can be reduced to half-space queries, and thus half-space queries constitute a rather general class of range queries.

4 Random Sampling

In this section, we discuss the technique of randomly sampling the database of points to approximate the number of points in a query range. The usefulness of random sampling methods in computational geometry has been recently demonstrated by Clarkson for a variety of problems (see [11, 12, 13, 14]). Random sampling for range queries was also used by Haussler & Welzl in [25].

The main idea behind the use of random sampling is that a sample may give useful approximate information about the sampled set. For example, consider the half-space counting problem on a database X of n points in E^d. When a query half-space h^* from H_d^* is introduced, we can pick a sample $Z \subset X$ consisting of m independently drawn points from the n database points. The idea is that the fraction of the m sample points that fall in the query range h^* is a good approximation for the fraction of the points of X that lie in the half-space h^*. Of course, the quality of the approximation improves as the size of the sample set Z increases.

To make the analysis simpler, we assume that $m << n$, that is, we pick a small fraction of the points of X. Suppose that r out of the n points of X lie in the range h^*, and define $p = r/n$. The probability that a randomly picked point from X falls in the range h^* is thus p. We define the random variable W to be the number of points of the random sample Z that fall in the range h^*. Since the m sample points are picked independently, and because we assume that $m << n$, the random variable W has a binomial distribution with probability p. The expected value of W is $\mu = mp = mr/n$ and the variance is $\sigma^2 = mp(1-p) = mr(n-r)/n^2$. Chebychev's inequality then gives us

$$\Pr\left\{\left|W - \frac{mr}{n}\right| > k\right\} < \frac{mp(1-p)}{k^2} ,$$

or equivalently that

$$\Pr\left\{\left|\frac{W}{m} - \frac{r}{n}\right| > \epsilon\right\} < \frac{mp(1-p)}{\epsilon^2 m^2} .$$

We can now get a bound for the probability that the fraction W/m is an estimate of r/n, to within an accuracy of ϵ

$$\Pr\left\{\left|\frac{W}{m}-\frac{r}{n}\right| \leq \epsilon\right\} \geq 1-\frac{1}{4\epsilon^2 m} .$$

The interpretation of this inequality is that for a sample of size $m \geq 1/(4\epsilon^2\delta)$, the fraction W/m approximates the fraction r/n to within an accuracy of ϵ, with probability at least $1-\delta$. Notice that the analysis is not limited to half-space counting queries; it applies to any range counting query in any range space, provided that we pick the random sample anew with every query. This discussion and analysis give rise to a data structure for approximate range counting that achieves $O(1)$ query time (actually $O(m)$ query time, but we assume that m is a constant), and uses $O(n)$ storage to store the n database points. There is no preprocessing required, and thus the preprocessing time for such a data structure is also $O(1)$. The main drawback of this data structure is that we have to pick a new random sample for each query. We would prefer to have the random sample picked once, and use the same sample to approximate successive range queries.

The problem of using one random sample for approximating several range queries has been studied in Statistics. Vapnik & Chervonenkis in [30] derived general conditions, under which several probabilities can be uniformly estimated using one random sample. The following definition of an ϵ-approximation was introduced by Vapnik & Chervonenkis.

Definition 4 Let $\mathcal{T} = (\mathcal{V}, \mathcal{R})$ be a range space and let X be a database. For any $0 \leq \epsilon \leq 1$ and $Z \subset X$, we say that Z is an *ϵ-approximation* of X (for $\mathcal{R}$), if for all $Y \in \mathcal{R}$ we have

$$\left|\frac{|X \cap Y|}{|X|} - \frac{|Z \cap Y|}{|Z|}\right| \leq \epsilon .$$

Vapnik & Chervonenkis also showed that ϵ-approximations can be constructed using random samples. They provided an upper bound on the size of a random sample needed to construct an ϵ-approximation of database X with probability at least $1-\delta$, for any range space with V-C dimension d. More precisely, they proved that if $\mathcal{T} = (\mathcal{V}, \mathcal{R})$ is a range space of V-C dimension d, X is a database, and ϵ and δ are real numbers, $0 < \epsilon, \delta \leq 1$, then a random sample Z of X of size m will be an ϵ-approximation of X for $\mathcal{R}$ with probability at least $1-\delta$, provided

$$m \geq \frac{c}{\epsilon^2}\left(d \lg\left(\frac{d}{\epsilon}\right) + \lg\left(\frac{1}{\delta}\right)\right) ,$$

for some positive real constant c.

From the result of Vapnik & Chervonenkis, it follows that for any database X, there is an ϵ-approximation Z of size at most $(c/\epsilon^2)(d \lg(d/\epsilon)) + 1$, which is independent of the size of X. For example, if X is a set of points in E^2, then there exists a 0.01-approximation Z for half-plane range queries, such that $|Z| = 2,525,039$. Similarly, there are ϵ-approximations of $O(1)$ size for all the range queries discussed in the introduction.

These ideas can be used to construct a data structure for range queries, that achieves $O(1)$ query time and uses only $O(1)$ storage. Furthermore, the preprocessing, which involves obtaining an ϵ-approximation by randomly sampling the database, can also be done in $O(1)$ expected time. This data structure, however, can only support approximate solutions for range queries. In addition, it can only be used for range counting queries and not for other kinds of queries (like reporting or minimizing). Finally, although the sample is of constant size, for most range spaces, the constants are quite large.

The technique of random sampling can be also employed to obtain data structures that enable exact solutions to range queries. This was demonstrated by Clarkson [12], who used random sampling to construct a search-tree based data structure for the point location problem in E^d. Another use of random sampling was later demonstrated by Haussler & Welzl in [25], to construct a space-partition-tree based data structure for half-space and simplicial range queries. Their scheme, which we describe in Section 6, uses the notion of an ϵ-net, which is related to the notion of an ϵ-approximation of [30].

Definition 5 Let $\mathcal{T} = (\mathcal{V}, \mathcal{R})$ be a range space, X a database, and $0 \leq \epsilon \leq 1$. Then $\mathcal{R}_{X,\epsilon}$ denotes the set of all $Y \in \mathcal{R}$ that contain strictly more than $\epsilon |X|$ points from X, that is, $|X \cap Y| / |X| > \epsilon$. A subset Z of X is an *ϵ-net* of X (for $\mathcal{R}$) if Z contains a point in each $Y \in \mathcal{R}_{X,\epsilon}$.

It is easily seen that every ϵ-approximation of X is also an ϵ-net of X. The converse, however, is not true. In general, the size of an ϵ-net can be much smaller then the size of an ϵ-approximation. Haussler & Welzl showed that ϵ-nets can also be constructed using random sampling. They provided an upper bound on the size of a random sample needed to construct an ϵ-net of database X with probability at least $1 - \delta$, for any range space with V-C dimension d. The random sample Z should be formed by m independent draws from X, where

$$m \geq \max\left\{ \frac{4}{\epsilon} \lg\left(\frac{2}{\delta}\right) \ , \ \frac{8d}{\epsilon} \lg\left(\frac{8d}{\epsilon}\right) \right\} .$$

From this discussion it follows that for any database X, there is an ϵ-net Z of size at most $(8d/\epsilon)\lg(8d/\epsilon)$. Notice that an ϵ-net of X is required to be a subset of X. Thus, for example, for any database X in E^d and for half-spaces as ranges, any 0-net must contain all the extreme points of X. However, if we lift the requirement that the points of the ϵ-net are a subset of X, there always exist $d+1$ points in E^d that contain X in the convex domain defined by them. These points, then, would constitute a 0-net of X of size $d+1$.

5 Search-Tree Tables

In this section, we describe the method of search-tree tables for range queries and demonstrate its use. A search-tree table is the underlying data structure of many algorithms that achieve polylogarithmic search time. This search time usually comes at the expense of polynomially large storage.

The main idea behind the search-tree tables method is that for a finite set of points (a database) and for certain types of range queries, there are only polynomially many possible solutions. Therefore, for certain kinds of range queries on a given database, one can initially store all the possible solutions in a table, and when a query range is introduced, the appropriate entry in the table can be located. The organization of these tables is usually based on search trees, which enables achieving polylogarithmic search time. The storage required by these data structures is polynominal in n, the size of the database. The preprocessing time is usually also polynomial in n.

In order to reduce the amount of storage, some algorithms, based on search-tree tables, do not store entire solutions in table entries. Rather, table entries contain partial information about the solution and pointers for further search in the table. These data structures, achieving smaller storage, are more appropriate for range reporting queries than for range counting queries. Recall that for range reporting queries, the search time is only one component of the query time, with the other component being the report time. The data structures described in this section will be analyzed in terms of their search time, ignoring the report time component. Recently, Chazelle [6] proposed a technique called *filtering search*, that reduces the total query time of certain range reporting queries, by attempting to match the search time and report time components of the query time.

The simplest example of search-tree tables is probably 1-dimensional binary search trees. The typical range query in 1 dimension is to determine the keys in some interval. Balanced binary search trees achieve logarithmic search time and require linear storage. Binary search trees do not contain entire solutions to range queries in their leaves, but still enable retrieving intervals. Threaded binary search trees are a more typical example of a data structure that stores partial solutions and pointers in leaves. There are several generalizations of the 1-dimensional search trees to higher dimensions. Multidimensional search trees were investigated mainly in the context of database queries (see [2, 3, 4, 19, 28, 29]). These data structures support orthogonal range queries in d dimensions. The storage is typically $O(n \lg^{d-1} n)$ with a search time of $O(\lg^d n)$. Notice, however, that these multidimensional search trees are dynamic, that is, they support insertions and deletions as well as orthogonal range queries.

The more general problem of half-space range queries has a natural representation as a search problem. There is a duality relationship between half-space range queries and point location problems. Using the Hough transformation (see Section 2), a set X of n points in E^d is mapped to a collection H_X of n hyperplanes from H_d. The problem of reporting the points of X that lie above some query hyperplane h transforms to the problem of reporting the hyperplanes of H_X that lie above the dual point p_h of h. If the arrangement $\mathcal{A}(H_X)$ of the hyperplanes H_X is given, then the latter problem can be solved by locating the cell in the arrangement $\mathcal{A}(H_X)$ containing p_h, thereby reducing the half-space range query problem to the point location problem.

The above reduction motivates a data structure for half-space range queries that represents the n database points by storing the arrangement of the dual n hyperplanes. The preprocessing involves computing the arrangement of the n dual hyperplanes and storing the solutions at the arrangement's cells. Each cell can contain a list (or a count) of the hyperplanes above and below it. This scheme requires $O(n^{d+1})$ storage for a data structure

for half-space range queries in E^d, since there are $O(n^d)$ cells in a general arrangement of n hyperplanes in E^d and each cell stores $O(n)$ information. An alternative data structure can be designed, where each cell only contains the "name" of a hyperplane bounding it from above (defining a facet of the cell), and a pointer to the cell on the other side of that facet. In this data structure it is required to first locate the cell containing the query point, and then to traverse a list of cells in the arrangement and report the appropriate hyperplanes. The search time accounts only for the time it takes to locate the first cell, while the time it takes to traverse the list is considered part of the report time. This scheme reduces the storage requirement to $O(n^d)$.

This discussion shows that efficient solutions for the point location problem in E^d also yield efficient solutions for half-space queries. In 2 dimensions, the point location problem can be solved in $O(\lg n)$ time by searching a planar subdivision tree (Kirkpatrick [26]). This forms the basis of the $O(n^3)$-storage, $O(\lg n)$-search-time data structure for half-plane queries described by Edelsbrunner & Kirkpatrick [20]. This scheme can be modified as described above to use only $O(n^2)$ storage. However, an optimal $O(n)$-storage, $O(\lg n)$-search-time data structure for the half-plane reporting problem was obtained by Chazelle, Guibas, & Lee [9]. Their method, however, only solves the half-plane reporting problem and is not extendible to other ranges in the plane.

A variety of other data structures for range queries, based on search-tree tables, appear in the literature. An $O(n(\lg n)^8(\lg\lg n)^4)$-storage, $O(\lg n)$-search-time data structure for the half-space reporting problem in E^3 was presented by Chazelle & Preparata in [10]. Their method again reduces the half-space range problem to a 3-dimensional point location problem. The small storage in their scheme is achieved by bounding the number of k-sets of an arbitrary point set in E^3. Improving the bound on the number of k-sets in E^3, Clarkson [14] was able to improve the storage requirement for the half-space reporting problem in E^3 to $O(n(\lg n)^2 \lg\lg n)$, while retaining the $O(\lg n)$ search time. In [20] Edelsbrunner, Kirkpatrick, and Maurer demonstrated an $O(n^7)$-storage, $O(\lg n)$-search-time data structure for triangular range queries in the plane.

A general data structure for the half-space reporting problem in d-dimensions, based on search-tree tables, was given by Cole & Yap [16]. We illustrate their technique in E^2, and comment about its generalization to E^d. Given a database X of n points in the plane, we draw a vertical line L to the right of the points in X. For any pair of points p, q from X, we compute the intersection point of the line through p and q with the base line L. These $\binom{n}{2}$ intersection points divide L into $\binom{n}{2}+1$ intervals with the following property. Consider rotating a test line P from slope $-\infty$ to slope ∞ about some fixed point r on L. This test line P meets the points of X in some order, which we associate with the intersection point r. Then, the orders of the points of X associated with any two points in a given interval on L are the same. This leads to the following data structure. With each of the $\binom{n}{2}+1$ intervals on L, we store the order of the points of X as seen from that interval. These $O(n^2)$ intervals are organized in a search tree, such that for any line h in E^2, the interval in which h intersects L can be found in $O(\lg n)$ time. When given a query half-plane h^*, we find the intersection interval of h with L and scan the list of points of X, stored with that interval, until we meet the first point of X that lies below h. This gives an $O(n^3)$-storage, $O(\lg n)$-search-time data structure for the half-plane reporting problem.

(Actually, the storage can be reduced to $O(n^2)$.) This data structure can be generalized to handle half-space reporting problems for E^d in $O(2^d \lg n)$ search time and $O(dn^{2^{d-1}})$ storage.

An improved data structure for half-space range queries, based on search-tree tables, is obtained from a new scheme of searching arrangements of hyperplanes in E^d, presented by Clarkson [12]. The data structure for the point location problem in an arrangement of hyperplanes is constructed using random sampling. The resulting algorithm achieves an $O(\lg n)$ query search time, and uses $O(n^{d+\epsilon})$ storage. The probabilistic construction of the data structure takes $O(n^{d+\epsilon})$ expected time. We next describe the essential ideas of the arrangement-searching algorithm.

In order to search the arrangement $\mathcal{A}(H)$ of a collection H of n hyperplanes, we use the triangulation $\Delta(\mathcal{A}(H))$, which is a recursive refinement of $\mathcal{A}(H)$ into simplicial regions. Rather then finding a cell C of $\mathcal{A}(H)$ containing the point, we find a simplex S which is a refinement of C. We search $\Delta(\mathcal{A}(H))$ by first searching $\Delta(\mathcal{A}(J))$, the triangulation of the arrangement of a small subset J of H. After locating the simplex S of $\Delta(\mathcal{A}(J))$ containing the query point, we find the subset H^* of the hyperplanes of H that intersect the simplex S, and continue the search with these hyperplanes. (This is a generalization of the scheme for point location in the plane presented by Kirkpatrick [26].) The construction of Clarkson's search tree is probabilistic. At each level of the tree, we pick a random sample of the current set of hyperplanes. For each simplex in the triangulation, we determine the set of hyperplanes intersecting it, and recursively build a search tree for that simplex with these hyperplanes. Clarkson showed that samples that generate a tree of logarithmic depth can be each found in $O(1)$ expected time.

A deterministic construction of the search-tree-table based data structure of Clarkson for the point location problem was recently presented by Chazelle & Friedman [8]. This data structure achieves an $O(\lg n)$ search time, uses $O(n^d)$ storage, and can be deterministically constructed in $O(n^{2+d(d+3)/2}/\lg n)$ time. Recently, a new algorithm for the half-space reporting problem in E^d was discovered by Clarkson [14]. This algorithm employs another data structure, based on search-tree tables, that achieves an $O(\lg n)$ search time, and uses $O(n^{\lfloor d/2 \rfloor + \epsilon})$ expected storage.

6 Space-Partition Trees

In this section, we describe the method of space-partition trees for range queries and demonstrate its use. A space-partition tree is the underlying data structure of many algorithms that use linear storage and achieve sublinear query time. The sublinear query time is achieved by using divide and conquer to search only part of the space-partition tree.

Space-partition trees can be informally described as follows. Given a database X of the range space $\mathcal{T} = (\mathcal{V}, \mathcal{R})$, we construct a rooted tree P with $|X|$ leaves and with a bounded number of children per node. Each node p of the tree P represents some region $reg(p)$ of the space. The root represents the whole space and the region represented by each internal

node in the tree is the disjoint union of the regions represented by its children. Thus, at every level of the tree P, the disjoint union of the regions represented by all the nodes at that level is the whole space. With every node p of P, we associate a set $set(p)$ of points of X. This set is simply the set of points of X that fall in the region represented by p (that is, $set(p) = X \cap reg(p)$). This guarantees that for an internal node p with children $p_1, p_2, \ldots, p_k$, we have $set(p) = set(p_1) \cup set(p_2) \cup \cdots \cup set(p_k)$, where the union is disjoint. This recursive partitioning the space is terminated when the set associated with a leaf p is a singleton, that is, $set(p) = \{x\}$, for some $x \in X$. In other words, the bottom level regions each contain a single element of X.

Space-partition trees can be used for solving range counting problems on a range space $\mathcal{T} = (\mathcal{V}, \mathcal{R})$. Suppose that P is a space-partition tree for a database X of $\mathcal{T}$. With any node p of P, we store the number of points in $set(p)$. To solve a range counting query for a range $Y \in \mathcal{R}$, we search P using the following divide and conquer scheme, starting at the root.

1. If $reg(p) \subset Y$, add $|set(p)|$ to the count and do not continue with p's children.
2. If $reg(p) \cap Y = \emptyset$, do not change the count and do not continue with p's children.
3. If $reg(p)$ *crosses* Y, that is, neither $reg(p) \subset Y$ nor $reg(p) \cap Y = \emptyset$, then continue the query recursively with p's children.

The usefulness of space-partition trees for range queries depends on a number of factors. First, we need to guarantee that a space-partition tree for range counting queries requires only linear storage, by storing only $O(1)$ information per node and having the number of nodes be linear in n. In particular, we do not need to store $set(p)$ itself at node p. Second, note that the query time depends on the number of nodes visited in the search; thus, to achieve sublinear query time for a range Y, we need to guarantee that only a fraction of the children of each node p are explored, that is, that the range Y crosses only a fraction of the regions represented by p's children. This property, however, does not have to hold for every single node of P; it is enough to guarantee it for all nodes of depth greater than or equal to some constant r. Third, for a node p and a range Y, we need to be able to determine quickly (in $O(1)$ time) whether $reg(p)$ is a subset of Y, whether $reg(p)$ and Y are disjoint, or whether Y crosses $reg(p)$.

The issues discussed above restrict the types of ranges for which range queries can be solved efficiently using space-partition trees. We described above how to solve range counting queries using a constant amount of information per node. Range reporting queries can be solved similarly by recursively visiting all the descendants of a node p for which $reg(p)$ is fully contained in the query range Y. The query time, however, for range reporting queries has the additional term of k, the number of points reported. The other variants of range queries mentioned in Section 3 can be solved similarly. We also observe that space-partition trees can only be used for range spaces of finite V-C dimension. In range spaces with infinite V-C dimension, there are arbitrarily large finite sets of points (databases) that do not have good space-partition trees. This observation is based on the fact that in range spaces with infinite V-C dimension there is no recursive partition of the space

that guarantees that arbitrary ranges cross only a small fraction of the partition's regions. Welzl [31] recently proved that the converse also holds, specifically, that range spaces with finite V-C dimension have good partition trees. Finally, we comment that the problem of determining the relationship between an arbitrary range Y and the region represented by a node p can be quite difficult in general.

We now present a variety of space-partition trees that were developed for answering certain kinds of range queries. The simplest space-partition trees for range queries are probably binary search trees. In 1 dimension, half-spaces, orthogonal ranges, circular ranges, simplex, and polyhedral range, all reduce to intervals. Queries of this type can be answered by exploring at most 1 of the 2 children of every node, leading to an $O(\lg n)$ query time with linear storage, achieved by a number of balanced binary search trees schemes.

One generalization of binary search trees to $d \geq 2$ dimensions are quad trees (see [5, 22, 28]). These trees recursively partition a region in E^d into 2^d subregions by bisecting the region along the d respective dimensions. It is important to note that quad trees do not necessarily give good space-partition trees for an arbitrary point set X in E^d. Whenever good space-partition quad trees exist, however, they are quite efficient for orthogonal range queries, achieving $O(n^{1-1/d})$ query time with linear storage. The $O(n^{1-1/d})$ query time follows from the observation that orthogonal ranges cross at most 2^{d-1} of the 2^d subregions represented by a quad-tree node.

The first construction of space-partition trees for half-plane and polygonal range queries in E^2 was presented by Willard [32]. Willard described a *J-way polygon tree*, which is a recursive partition of the plane by J lines into $2J$ polygonal regions, each containing approximately $n/(2J)$ of the n database points. Willard's polygon tree has the property that any line in the plane intersects (crosses) at most $J+1$ of the $2J$ subregions represented by any node in the space-partition tree. In addition, for any polygon in the plane, there is some depth r of the tree, such that the polygon intersects at most $J+1$ of the $2J$ subregions represented by any node at depth greater than or equal to r. These properties lead to an $O(n^{\log_{2J}(J+1)})$ query time with linear storage for half-plane and polygonal range queries. This query time is minimized for $J = 3$, for which the 3-way polygon tree achieves $O(n^{\lg_6 4}) = O(n^{0.774})$ query time. The preprocessing time for constructing a 3-way polygon tree is $O(n^2)$.

Extending Willard's ideas for the 3-dimensional case, Yao [35] showed that a space-partition tree can be constructed for any point set in E^3. Yao demonstrated an *octant-tree* for half-space and polyhedral range queries that uses linear storage and has an $O(n^{0.98})$ query time. The octant-tree is a recursive partition of a region in E^3 by 3 planes into 8 subregions of dimension 3 and a few other subregions of lower dimension. Each of the 8 subregions of dimension 3 is guaranteed to contain at least $1/24$ of the points of the parent region. Any plane in E^3 intersects at most 7 of the 8 subregions represented by a node in the octant-tree. When the lower dimensional subregions are also taken into account, the query time is shown to be $O(n^{0.98})$. Following Yao's result, Dobkin, Edelsbrunner, and Yao [18] showed that using the same ideas, the query time can be improved to $O(n^{\log_8 7}) = O(n^{0.936})$. The preprocessing time of Yao's approach was $O(n^4)$, which was later improved to $O(n^2 \lg^{10} n)$ by Cole, Sharir, and Yap [15].

The first general scheme of space-partition trees for any dimensional Euclidean space E^d was presented by Yao & Yao in [36]. They treated a variety of range queries and some other optimization queries under one class of *geometric generic queries*, and demonstrate a general scheme of solving generic queries in linear storage and sublinear query time. They showed that any finite point set X in E^d has a regular space-partition tree P of degree 2^d with the following property. For any internal node p of P and any child q of p we have $|set(q)| \leq (1/2^d)\,|set(p)|$, and any hyperplane h in E^d intersects at most $2^d - 1$ subregions of children of p. This leads to a linear storage, sublinear query-time scheme for half-space range queries in any dimensional Euclidean space. The query time of Yao & Yao's scheme is $O(n^{\lg(2^d-1)/d})$ for half-space and polyhedral range queries in E^d.

Unfortunately, Yao & Yao's scheme is not constructive. But, although Yao & Yao do not provide a practical scheme for constructing the balanced space-partition tree in E^d, their result is of primary importance, as it demonstrates that such balanced space-partition trees exist for any finite point set in E^d.

The best known query-time bounds for linear-storage data structures for half-space and simplicial range counting queries in E^d, for $d \geq 2$, were presented by Haussler & Welzl [25]. (A better query-time for E^2 was recently given by Welzl [31].) Their construction of space-partition trees uses random sampling and is based on the notion of ϵ-nets introduced in Section 3. Their scheme achieves $O(n^\alpha)$ query time, for $\alpha > \frac{d(d-1)}{d(d-1)+1}$. The construction of the space-partition tree is probabilistic and takes $O(n \lg n)$ expected time.

The partition trees of Haussler & Welzl, called (ϵ, v)-partition trees, are a variant of the space-partition trees described above. The main difference is that in (ϵ, v)-partition trees, the region represented by an internal node p is not divided into subregions containing approximately the same number of points. Rather, the subdivision of the region $reg(p)$ in (ϵ, v)-partition trees is determined as follows. A subset $points(p)$ of $v > d$ points from $set(p)$ is selected, and the $O(v^d)$ hyperplanes, each passing through exactly d of these v points, are computed. The arrangement of these $O(v^d)$ hyperplanes is determined and stored at node p as $arr(p)$. For each cell f in $arr(p)$ with $f \cap set(p) \neq \emptyset$, there is a child p_f with $reg(p_f) = f \cap reg(p)$. When the number of points in $set(p)$ is at most v, the recursive refinement terminates, that is, p is a leaf node.

In an (ϵ, v)-partition tree, the subdivision of the region represented by an internal node p depends on the selection of the v points from $set(p)$. A desired subdivision has the following property: for any hyperplane h in E^d, the total number of points of $set(p)$ in all cells of $arr(p)$ intersected by h, is at most $\epsilon\,|set(p)|$. This condition is immediately satisfied for $\epsilon = 1$, but we are interested in smaller values of ϵ. Intuitively, the smaller the value of ϵ is, the smaller the number of children of an internal node p that need be explored. The existence of (ϵ, v)-partition trees for any value of $0 < \epsilon < 1$ is based on the existence of ϵ-nets for some specific ranges (namely, $(d+1)$-corridors) of X.

The bounds of [25] are not optimal. Recently, it was shown by Welzl [31] that half-plane and triangular range queries in E^2 can be solved with linear storage and $O(\sqrt{n} \lg^3 n)$ query time. Welzl's algorithm also employs a space-partition tree based on ϵ-nets. These space-time bounds for triangular ranges are optimal up to a polylog factor (see Section 7 for corresponding lower bounds).

7 Lower Bounds

In this section we present some lower bounds on the query time and the storage requirements of various types of range queries.

Dynamic Data Structures for Orthogonal Range Queries. Investigating the complexity of dynamic data structures that support range queries as well as insertion and deletion, Fredman [23] presented some lower bounds for orthogonal range queries. He showed that the time complexity of performing a sequence of n intermixed manipulations with d-dimensional keys is $\Omega(n \lg^d n)$. This lower bound is tight for dynamic data structures for orthogonal range queries as was shown by Fredman & Bentley [3] and by Lueker [29].

Dynamic Data Structures for Half-plane, Circular, and Parabolic Ranges in E^2. Generalizing the above lower bounds to half-plane ranges, circular ranges, and parabolic ranges, Fredman showed in [24] an $\Omega(n^{4/3})$ lower bound on the worst case time complexity of processing a sequence of n manipulations. These bounds show that the orthogonal range queries are intrinsically much easier the the latter range queries.

Static Data Structures for Circular Ranges in E^2 and Half-Space Ranges in E^3. In [34] Yao showed that any static data structure for circular ranges in E^2, that uses $O(n)$ storage, must have an $\Omega(n^\epsilon)$ query time, for some $\epsilon > 0$. Since circular range queries in E^2 can be reduced to half-space range queries in E^3 (see Yao [35]), the same bound applies for half-space range queries in E^3.

Static Data Structures for Simplex Counting Queries. In [7] Chazelle proved a family of lower bounds involving the query time and the storage complexities of simplex counting range queries. For dimension $d = 2$, if the storage complexity of an algorithm is $O(m)$, then the query time complexity is shown to be $\Omega(n/\sqrt{m})$. More generally, in dimension $d > 2$, the query time complexity is $\Omega((n/\lg n)/m^{1/d})$. These bounds hold with high probability for a random point set, and are thus valid in the worst case as well as in the average case.

These bounds imply, for instance, that if the storage is restricted to be linear, then the query time for $d = 2$ is $\Omega(\sqrt{n})$. For $d > 2$ dimensions, the query time is $\Omega(n^{1-1/d}/\lg n)$. On the other hand, for the query time to be polylogarithmic in n, the storage has to be $\Omega(n^{d-\epsilon})$ in E^d, for $d \geq 2$. It is important to emphasize that these bounds are only for the simplex counting problem.

Static Data Structures for Simplex Reporting Queries. The bounds for the simplex reporting problem are different than those for the simplex counting problem, since the query time has two components, the search time and the report time. For a slightly weaker machine model, Chazelle [7] proved that if the query time is $O(\lg^b n + k)$, where k is the number of points reported and $b \geq 1$, then the storage requirement is $\Omega(n^{d-\epsilon})$, where d is the dimension of the space.

This lower bound is quite surprising, since it implies that in E^2, a polylogarithmic query time for the simplex reporting problem requires $\Omega(n^{2-\epsilon})$ storage. For the half-plane reporting problem, however, there is an algorithm by Chazelle, Guibas, and Lee [9] that uses $O(n)$ storage and its query time is $O(\lg n + k)$. This demonstrates that simplex range queries are intrinsically harder than half-space range queries.

8 Conclusions

In this section we summarize the three methods for range queries and conclude with some possible research directions.

Summary of Methods. The three methods, random sampling, search-tree tables, and space-partition trees have different tradeoffs between query time, storage requirement, preprocessing, and their applicability to different types of range queries. When only approximate solutions are required, random sampling seems to be the method of choice. Furthermore, random sampling applies to a large variety of ranges, with relatively little preprocessing. Both search-tree tables and space-partition trees can be used for exact counting and reporting queries. Search-tree tables, when applicable, are a better choice for reporting queries, since their search time is polylogarithmic. Space-partition trees, however, achieve linear storage and sublinear query time, which may be the desired tradeoff for a variety of range counting queries. The following table summarizes the important characteristics of the three methods. The constants in the table satisfy $a, b, c, p, q \geq 1$ and $0 < \alpha < 1$.

	Random Sampling	Search-Tree Tables	Space-Partition Trees
Query time	$O(1)$	$O(\lg^a n)$	$O(n^\alpha)$
Storage	$O(1)$	$O(n^b)$	$O(n)$
Preprocessing	$O(1)$	$O(n^p)$	$O(n^q)$
Exact answer	no	yes	yes
Range spaces	finite V-C dimension	$O(n^c)$ solutions	finite V-C dimension
Query type	counting	counting/reporting	counting/reporting

The three methods are not completely unrelated. Both search-tree tables and space-partition trees can be considered as possible generalizations of 1-dimensional binary trees. A 1-dimensional binary tree can be thought of as a search tree, with solutions stored at its leaves, which is the underlying scheme of search-tree tables. Alternatively, a 1-dimensional binary tree can be thought of as a recursive partition of the 1-dimensional space into two half spaces, which is the underlying scheme of space-partition trees. In addition, it is sometimes possible to combine two methods, as was demonstrated for random sampling and search-tree tables, and for random sampling and space-partition trees. In these cases, random sampling was used in the construction of the data structure but not in processing the queries.

Tight Upper and Lower Bounds. Almost all the types of range queries mentioned in this paper do not have matching upper and lower bounds. The bounds are tight only for orthogonal range queries. Tighter bounds are certainly of both theoretical and practical significance.

Dynamic Data Structures for Range Queries. With the exception of orthogonal range queries, no optimal dynamic data structures are known for general range queries in d dimensions. Since dynamic data structures are of primary significance, the design of efficient dynamic data structures for range queries is an important direction of research.

Reductions of Range Queries. As was mentioned in the introduction, some types of range queries can be reduced to others. These reductions usually involve transformations of objects from a d-dimensional Euclidean space to an Euclidean space of higher dimension. Other such reductions and relations between different types of ranges are of interest.

Acknowledgements

This survey paper is an outcome of the area exam of the author. The author would like to thank the committee members Ron Rivest, Albert Meyer, and Carl Hewitt for contributing to the understanding of the problems involved and for several suggestions. Thanks also to Alok Aggarwal, Dina Kravets, Charles Leiserson, and James Park for general comments and suggestions regarding this paper.

References

[1] N. Alon, D. Haussler, and E. Welzl, "Partitioning and geometric embedding of range spaces of finite Vapnik-Chervonenkis dimension," *3rd Annual Symposium on Computational Geometry,* ACM, 1987, pp. 331–340.

[2] J. L. Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM,* Vol. 18, No. 9, 1975, pp. 509–517.

[3] J. L. Bentley and J. Friedman, "Algorithms and data structures for range searching," *Computer Science and Statistics: 11th Annual Symposium on Interface,* 1978, pp. 297–307.

[4] J. L. Bentley and H. A. Maurer, "Efficient worse-case data structures for range searching," *Acta Informatica,* Vol. 13, 1980, pp. 155–168.

[5] J. L. Bentley and D. F. Stanat, "Analysis of range searches in quad trees," *Information Processing Letters,* Vol. 3, 1975, pp. 170–173.

[6] B. Chazelle, "Filtering search: a new approach to query-answering," *24th Annual Symposium on Foundations of Computer Science,* IEEE, 1983, pp. 122–132.

[7] B. Chazelle, "Polytope range searching and integral geometry," *28th Annual Symposium on Foundations of Computer Science,* IEEE, 1987, pp. 1–10.

[8] B. Chazelle and J. Friedman, "A deterministic view of random sampling and its use in geometry," *29th Annual Symposium on Foundations of Computer Science,* IEEE, 1988, pp. 539–549.

[9] B. Chazelle, L. Guibas, and D. T. Lee, "The power of geometric duality," *24th Annual Symposium on Foundations of Computer Science,* IEEE, 1983, pp. 217–225.

[10] B. Chazelle and F. P. Preparata, "Halfspace range search: an algorithmic application of k-sets," *Discrete and Computational Geometry,* Vol. 1, 1986, pp. 83–93.

[11] K. L. Clarkson, "A probabilistic algorithm for the post office problem," *17th Annual Symposium on Theory of Computing,* ACM, 1985, pp. 175–184.

[12] K. L. Clarkson, "Further applications of random sampling to computational geometry," *18th Annual Symposium on Theory of Computing,* ACM, 1986, pp. 414–423.

[13] K. L. Clarkson, "New applications of random sampling in computational geometry," *Discrete and Computational Geometry,* Vol. 2, 1987, pp. 195–222.

[14] K. L. Clarkson, "Applications of random sampling in computational geometry, II," *4th Annual Symposium on Computational Geometry,* ACM, 1988, pp. 1–11.

[15] R. Cole, M. Sharir, and C. K. Yap, "On k-hulls and related problems," *16th Annual Symposium on Theory of Computing,* ACM, 1984, pp. 154–166.

[16] R. Cole and C. K. Yap, "Geometric retrieval problems," *24th Annual Symposium on Foundations of Computer Science,* IEEE, 1983, pp. 112–121.

[17] D. P. Dobkin and H. Edelsbrunner, "Space searching for intersecting objects," *25th Annual Symposium on Foundations of Computer Science,* IEEE, 1984, pp. 387–391.

[18] D. P. Dobkin, H. Edelsbrunner, and F. Yao, "A 3-space partition and its applications," manuscript.

[19] D. P. Dobkin and R. Lipton, "Multidimensional searching problems," *SIAM Journal of Computing,* Vol. 5, No. 2, May 1976, pp. 181–186.

[20] H. Edelsbrunner, D. G. Kirkpatrick, and H. A. Maurer, "Polygonal intersection searching," *Information Processing Letters,* Vol. 14, 1982, pp. 74–79.

[21] H. Edelsbrunner and E. Welzl, "Halfplanar range search in linear space and $O(n^{0.695})$ query time," *Information Processing Letters,* Vol. 23, 1986, pp. 289–293.

[22] R. A. Finkel and J. L. Bentley, "Quad trees: a data structure for retrieval on composite keys," *Acta Informatica,* Vol. 4, 1974, pp. 1–9.

[23] M. L. Fredman, "A lower bounds on the complexity of orthogonal range queries," *Journal of the ACM,* Vol. 28, 1981, pp. 696–705.

[24] M. L. Fredman, "Lower bounds on the complexity of some optimal data structures," *SIAM Journal of Computing,* Vol. 10, No. 1, February 1981, pp. 1–10.

[25] D. Haussler and E. Welzl, "ϵ-nets and simplex range queries," *Discrete and Computational Geometry,* Vol. 2, 1987, pp. 127–151.

[26] D. G. Kirkpatrick, "Optimal search in planar subdivisions," *SIAM Journal of Computing,* Vol. 12, No. 1, February 1983, pp. 28–35.

[27] D. T. Lee, "On k-nearest neighbor Voronoi diagrams in the plane," *IEEE Transactions on Computers,* Vol. C-31, No. 6, June 1982, pp. 478–487.

[28] 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, 1977, pp. 23–29.

[29] G. S. Lueker, "A data structure for orthogonal range queries," *19th Annual Symposium on Foundations of Computer Science,* IEEE, 1978, pp. 28–34.

[30] V. N. Vapnik and A. Y. Chervonenkis, "On the uniform convergence of relative frequencies of events to their probabilities," *Theory of Probability and its Applications,* Vol. 16, 1971, pp. 264–280.

[31] E. Welzl, "Partition trees for triangle counting and other range searching problems," *4th Annual Symposium on Computational Geometry,* ACM, 1988, pp. 23–33..

[32] D. E. Willard, "Polygon retrieval," *SIAM Journal of Computing,* Vol. 11, No. 1, February 1982, pp. 149–165.

[33] A. C. Yao, "On constructing minimum spanning trees in k-dimensional spaces and related problems," *SIAM Journal of Computing,* Vol. 11, No. 4, November 1982, pp. 721–736.

[34] A. C. Yao, "Space-time tradeoff for answering range queries," *14th Annual Symposium on Theory of Computing,* ACM, 1982, pp. 128–136.

[35] F. Yao, "A 3-space partition and its applications," *15th Annual Symposium on Theory of Computing,* ACM, 1983, pp. 258–263.

[36] A. C. Yao and F. Yao, "A general approach to d-dimensional geometric queries," *17th Annual Symposium on Theory of Computing,* ACM, 1985, pp. 163–168.