31 RANGE SEARCHING

Pankaj K. Agarwal

INTRODUCTION

Range searching is one of the central problems in computational geometry because it arises in many applications and a variety of geometric problems can be formulated as range-searching problems. A typical range-searching problem has the following form. Let S be a set of n points in $\mathbb{R}^d$, and let $\mathcal{R}$ be a family of subsets of $\mathbb{R}^d$; elements of $\mathcal{R}$ are called ***ranges***. *We wish to preprocess S into a data structure, so that for a query range R, the points in $S \cap R$ can be reported or counted efficiently.* Typical examples of ranges include rectangles, halfspaces, simplices, and balls. If we are only interested in answering a single query, it can be done in linear time, using linear space, by simply checking each point of S whether it lies in the query range. However, most of the applications call for querying the same set S several times (perhaps with periodic insertions and deletions), in which case we would like to answer a query faster by preprocessing S into a data structure.

Range counting and range reporting are just two instances of range-searching queries. Other examples include *emptiness queries*, in which one wants to determine whether $S \cap \gamma = \emptyset$, and *optimization queries*, in which one wants to choose a point with certain property (e.g., a point in γ with the largest x_1-coordinate). In order to encompass all different types of range-searching queries, a general range-searching problem can be defined as follows.

Let $(\mathbf{S}, +)$ be a commutative semigroup. For each point $p \in S$, we assign a weight $w(p) \in \mathbf{S}$. For any subset $S' \subseteq S$, let $w(S') = \sum_{p \in S'} w(S)$, where addition is taken over the semigroup. For a query range $\gamma \in \mathcal{R}$, we wish to compute $w(S \cap \gamma)$. For example, counting queries can be answered by choosing the semigroup to be $(\mathbb{Z}, +)$, where $+$ denotes standard integer addition, and setting $w(p) = 1$ for every $p \in S$; emptiness queries by choosing the semigroup to be $(\{0, 1\}, \vee)$ and setting $w(p) = 1$; reporting queries by choosing the semigroup to be $(2^S, \cup)$ and setting $w(p) = \{p\}$; and optimization queries by choosing the semigroup to be $(\mathbb{R}, \max)$ and choosing $w(p)$ to be, for example, the x_1-coordinate of p.

We can, in fact, define a more general (decomposable) *geometric searching* problem. Let S be a set of *objects* in $\mathbb{R}^d$ (e.g., points, hyperplanes, balls, or simplices), $(\mathbf{S}, +)$ a commutative semigroup, $w : S \to \mathbf{S}$ a weight function, $\mathcal{R}$ a set of ranges, and $\Diamond \subseteq S \times \mathcal{R}$ a "spatial" relation between objects and ranges. Then for a range $\gamma \in \mathcal{R}$, we want to compute $\sum_{p \Diamond \gamma} w(p)$. Range searching is a special case of this general searching problem in which S is a set of points in $\mathbb{R}^d$ and $\Diamond = \in$. Another widely studied searching problem is *intersection searching*, where $p \Diamond \gamma$ if p intersects γ. As we will see below, range-searching data structures are useful for many other geometric searching problems.

The performance of a data structure is measured by the time spent in answering a query, called the *query time*, by the *size* of the data structure, and by the time constructed in the data structure, called the *preprocessing time*. Since the data structure is constructed only once, its query time and size are generally more

0-8493-8524-5/97/\$0.00+\$.50

important than its preprocessing time. If a data structure supports insertion and deletion operations, its *update time* is also relevant. We should remark that the query time of a range-reporting query on any reasonable machine depends on the output size, so the query time for a range-reporting query consists of two parts — *search time*, which depends only on n and d, and *reporting time*, which depends on n, d, and the output size. Throughout this survey paper we will use k to denote the output size.

We assume that d is a small fixed constant, and that big-Oh and big-Omega notation hide constants depending on d. The dependence on d of the performance of almost all the data structures mentioned in this survey is exponential, which makes them unsuitable in practice for large values of d.

The size of any range-searching data structure is at least linear, since it has to store each point (or its weight) at least once, and the query time in any reasonable model of computation such as pointer machines, RAMs, or algebraic decision trees is $\Omega(\log n)$ even when $d = 1$. Therefore, we would like to develop a linear-size data structure with logarithmic query time. Although near-linear-size data structures are known for orthogonal range searching in any fixed dimension that can answer a query in polylogarithmic time, no similar bounds are known for range searching with more complex ranges such as simplices or disks. In such cases, we seek a tradeoff between the query time and the size of the data structure — How fast can a query be answered using $O(n\text{polylog}(n))$ space, how much space is required to answer a query in $O(\text{polylog}(n))$ time, and what kind of tradeoff between the size and the query time can be achieved?

This chapter is organized as follows. In Section 31.1 we describe various models of computation that are used for range searching. In Section 31.2 we review the orthogonal range-searching data structures, and in Section 31.3 we review simplex range-searching data structures. Section 31.4 surveys other variants and extensions of range searching, including multi-level data structures and kinetic range searching. In Section 31.5, we study intersection-searching problems, which can be regarded as a generalization of range searching. Finally, Section 31.6 deals with a few optimization queries.

31.1 MODELS OF COMPUTATION

Most geometric algorithms and data structures are implicitly described in the familiar *random access machine* (RAM) model, or the ***real RAM*** model. In the traditional RAM model, memory cells can contain arbitrary $(\log n)$-bit integers, which can be added, multiplied, subtracted, divided (computing $\lfloor x/y \rfloor$), compared, and used as pointers to other memory cells in constant time. In a real RAM, we also allow memory cells to store arbitrary real numbers (such as coordinates of points). We allow constant-time arithmetic on and comparisons between real numbers, but we do not allow conversion between integers and reals. In the case of range searching over a semigroup other than the integers, we also allow memory cells to contain arbitrary values from the semigroup, but only the semigroup-addition operations can be performed on them.

Many range-searching data structures are described in the more restrictive ***pointer-machine*** model. The main difference between RAM and pointer-machine models is that on a pointer machine, a memory cell can be accessed only through a series of pointers, while in the RAM model, any memory cell can be accessed in constant time. In the basic pointer-machine model, a data structure is a directed graph with outdegree 2; each node is associated with a label, which is an integer between 0 and n. Nonzero labels are indices of the points in S, and the nodes with label 0 store auxiliary information. The query algorithm traverses a portion of the graph and for each point in the query range it identifies at least one node that stores the index of that point. Chazelle [23] defines several generalizations of the pointer-machine model that are more appropriate for answering counting and semigroup queries. In Chazelle's generalized pointer-machine models, nodes are labeled with arbitrary $O(\log n)$-bit integers. In addition to traversing edges in the graph, the query algorithm is also allowed to perform various arithmetic operations on these integers. An *elementary* pointer machine (called EPM) can perform addition and comparisons between integers; an *arithmetic* pointer machine (called APM) can perform subtraction, multiplication, integer division, and shifting ($x \mapsto 2^x$).

If the input is too large to fit into main memory, then the data structure must be stored in secondary memory — on disk, for example — and portions of it must be moved into main memory when needed to answer a query. In this case the bottleneck in query and preprocessing time is the time spent in transferring data between main and secondary memory. A commonly used model is the standard *two-level* memory model, in which one assumes that data is stored in secondary memory in blocks of size B, where B is a parameter. Each access to secondary memory transfers one block (i.e., B words), and we count this as one input/output (I/O) operation. The size of a data structure is the number of blocks required to store it in secondary memory, and the query (resp. preprocessing) time is defined as the number of I/O operations required to answer a query (resp. to construct the structure). Under this model, the size of any data structure is at least n/B, and the range-reporting query time is at least $\log_B n + k/B$. There have been various extensions of this model, including the so-called *cache-oblivious* model in which one does not know the value of B and the goal is to minimize the I/O efficiency as well as the total work performed.

Most lower bounds, and a few upper bounds, are described in the so-called ***semigroup arithmetic model***, which was originally introduced by Fredman [45] and refined by Yao [81]. In the semigroup arithmetic model, a data structure can be *informally* regarded as a set of precomputed partial sums in the underlying semigroup. The size of the data structure is the number of sums stored, and the query time is the minimum number of semigroup operations required (on the precomputed sums) to compute the answer to a query. The query time ignores the cost of various auxiliary operations, including the cost of determining which of the precomputed sums should be added to answer a query. Unlike the pointer-machine model, the semigroup model allows immediate access, at no cost, to any precomputed sum.

The informal model we have just described is much too powerful. For example, in this semigroup model, the optimal data structure for range-counting queries consists of the $n + 1$ integers $0, 1, \ldots, n$. To answer a counting query, we simply return the correct answer; since no additions are required, we can answer queries in zero "time", using a "data structure" of only linear size! We need the notion of a *faithful* semigroup to circumvent this problem. A commutative semigroup $(\mathbf{S}, +)$ is faithful if for each $n > 0$, for any sets of indices $I, J \subseteq \{1, \ldots, n\}$ where $I \neq J$, and for every sequence of positive integers α_i, β_j ($i \in I, j \in J$), there are semigroup values $s_1, s_2, \ldots, s_n \in \mathbf{S}$ such that $\sum_{i \in I} \alpha_i s_i \neq \sum_{j \in J} \beta_j s_j$. For example, $(\mathbb{Z}, +)$, $(\mathbb{R}, \min)$, $(\mathbb{N}, \gcd)$, and $(\{0,1\}, \vee)$ are faithful, but $(\{0,1\}, + \bmod 2)$ is not faithful.

Let $S = \{p_1, p_2, \ldots, p_n\}$ be a set of objects, $\mathbf{S}$ a faithful semigroup, $\mathcal{R}$ a set of ranges, and $\Diamond$ a relation between objects and ranges. (Recall that in the standard range-searching problem, the objects in S are points, and $\Diamond$ is containment.) Let $x_1, x_2, \ldots, x_n$ be a set of n variables over $\mathbf{S}$, each corresponding to a point in S. A *generator* $g(x_1, \ldots, x_n)$ is a linear form $\sum_{i=1}^{n} \alpha_i x_i$, where α_i's are non-negative integers, not all zero. (In practice, the coefficients α_i are either 0 or 1.) A *storage scheme* for $(S, \mathbf{S}, \mathcal{R}, \Diamond)$ is a collection of generators $\{g_1, g_2, \ldots, g_s\}$ with the following property: For any query range $\gamma \in \mathcal{R}$, there is a set of indices $I_\gamma \subseteq \{1, 2, \ldots, s\}$ and a set of labeled nonnegative integers $\{\beta_i \mid i \in I_\gamma\}$ such that the linear forms

$$\sum_{p_i \Diamond \gamma} x_i \quad \text{and} \quad \sum_{i \in I_\gamma} \beta_i g_i$$

are identically equal. In other words, the equation

$$\sum_{p_i \Diamond \gamma} w(p_i) = \sum_{i \in I_\gamma} \beta_i g_i(w(p_1), w(p_2), \ldots, w(p_n))$$

holds for *any* weight function $w : S \to \mathbf{S}$. (Again, in practice, $\beta_i = 1$ for all $i \in I_\gamma$.) The size of the smallest such set I_γ is the query time for γ; the time to actually choose the indices I_γ is ignored. The space used by the storage scheme is measured by the number of generators. There is no notion of preprocessing time in this model.

The semigroup model is formulated slightly differently for offline range-searching problems. Here we are given a set of weighted points S and a finite set of query ranges $\mathcal{R}$, and we want to compute the total weight of the points in each query range. This is equivalent to computing the product Aw, where A is the incidence matrix of the points and ranges, and w is the vector of weights. In the offline semigroup model, introduced by Chazelle [28, 30], an algorithm can be described as a circuit with one input for every point and one output for every query range, where every gate performs a binary semigroup addition. The running time of the algorithm is the total number of gates.

A serious weakness of the semigroup model is that it does not allow subtractions even if the weights of points belong to a group. Therefore, we will also consider the ***group model***, in which both additions and subtractions are allowed [29].

Almost all geometric range-searching data structures are constructed by subdividing space into several regions with nice properties and recursively constructing a data structure for each region. Range queries are answered with such a data structure by performing a depth-first search through the resulting recursive space partition. The ***partition graph*** model, recently introduced by Erickson [38, 39], formalizes this divide-and-conquer approach, at least for simplex range searching data structures. The partition graph model can be used to study the complexity of emptiness queries, unlike the semigroup arithmetic and pointer machine models, in which such queries are trivial.

We conclude this section by noting that most of the range-searching data structures discussed in this paper (halfspace range-reporting data structures being a notable exception) are based on the following general scheme. Given a point set S, the structure precomputes a family $\mathcal{F} = \mathcal{F}(S)$ of *canonical subsets* of S and store the weight $w(C) = \sum_{p \in C} w(p)$ of each canonical subset $C \in \mathcal{F}$. For a query range γ, the query procedure determines a partition $\mathcal{C}_\gamma = \mathcal{C}(S, \gamma) \subseteq \mathcal{F}$ of $S \cap \gamma$ and adds the weights of the subsets in $\mathcal{C}_\gamma$ to compute $w(S \cap \gamma)$. We will refer to such a data structure as a ***decomposition scheme***.

There is a close connection between the decomposition schemes and the storage schemes of the semigroup arithmetic model described earlier. Each canonical subset $C = \{p_i \mid i \in I\} \in \mathcal{F}$, where $I \subseteq \{1, 2, \ldots, n\}$, corresponds to the generator $\sum_{i \in I} x_i$. How exactly the weights of canonical subsets are stored and how $\mathcal{C}_\gamma$ is computed depends on the model of computation and on the specific range-searching problem. In the semigroup (or group) arithmetic model, the query time depends only on the number of canonical subsets in $\mathcal{C}_\gamma$, regardless of how they are computed, so the weights of canonical subsets can be stored in an arbitrary manner. In more realistic models of computation, however, some additional structure must be imposed on the decomposition scheme in order to efficiently compute $\mathcal{C}_\gamma$. In a *hierarchical* decomposition scheme, the weights are stored in a tree T. Each node v of T is associated with a canonical subset $C_v \in \mathcal{F}$, and the children of v are associated with subsets of C_v. Besides the weight of C_v, some auxiliary information is also stored at v, which is used to determine whether $C_v \in \mathcal{C}_\gamma$ for a query range γ. If the weight of each canonical subset can be stored in $O(1)$ memory cells and if we can determine in $O(1)$ time whether $C_w \in \mathcal{C}_\gamma$ where w is a descendent of a given node v, we call the hierarchical decomposition scheme *efficient*. The total size of an efficient decomposition scheme is simply $O(|\mathcal{F}|)$. For range-reporting queries, in which the "weight" of a canonical subset is the set itself, the size of the data structure is reduced to $O(|\mathcal{F}|)$ by storing the canonical subsets implicitly. Finally, let $r > 1$ be a parameter, and set $\mathcal{F}_i = \{C \in \mathcal{F} \mid r^{i-1} \leq |C| \leq r^i\}$. We call a hierarchical decomposition scheme *r-convergent* if there exist constants $\alpha \geq 1$ and $\beta > 0$ so that the degree of every node in T is $O(r^\alpha)$ and for all $i \geq 1$, $|\mathcal{F}_i| = O((n/r^i)^\alpha)$ and, for all query ranges γ, $|\mathcal{C}_\gamma \cap \mathcal{F}_i| = O((n/r^i)^\beta)$, i.e., the number of canonical subsets in the data structure and in any query output decreases exponentially with their size. We will see below in Section 31.4 that r-convergent hierarchical decomposition schemes can be cascaded together to construct multi-level structures that answer complex geometric queries.

To compute $\sum_{p_i \in \gamma} w(p_i)$ for a query range γ using a hierarchical decomposition scheme T, a query procedure performs a depth-first search on T, starting from its root. At each node v, using the auxiliary information stored at v, the procedure determines whether γ contains C_v, whether γ intersects C_v but does not contain C_v, or whether γ is disjoint from C_v. If γ contains C_v, then C_v is added to $\mathcal{C}_\gamma$ (rather, the weight of C_v is added to a running counter). Otherwise, if γ intersects C_v, the query procedure identifies a subset of children of v, say $\{w_1, \ldots, w_a\}$, so that the canonical subsets $C_{w_i} \cap \gamma$, for $1 \leq i \leq a$, form a partition of $C_v \cap \gamma$. Then the procedure searches each w_i recursively. The total query time is $O(\log n + |\mathcal{C}_\gamma|)$, provided constant time is spent at each node visited.

31.2 ORTHOGONAL RANGE SEARCHING

In d-dimensional orthogonal range searching, the ranges are d-rectangles, each of the form $\prod_{i=1}^{d} [a_i, b_i]$ where $a_i, b_i \in \mathbb{R}$. This is an abstraction of ***multi-key searching***. For example, the points of S may correspond to employees of a company, each coordinate corresponding to a key such as age, salary, experience, etc. Queries of the form, e.g., "report all employees between the ages of 30 and 40 who earn more than \$30,000 and who have worked for more than 5 years," can be formulated as orthogonal range-reporting queries. Because of its numerous applications, orthogonal range searching has been studied extensively. In this section we review recent data structures and lower bounds.

UPPER BOUNDS

Most orthogonal range-searching data structures are based on ***range trees***, introduced by Bentley [17]. For a set S of n points in $\mathbb{R}^2$, the range tree T of S is a minimum-height binary tree with n leaves whose ith leftmost leaf stores the point of S with the ith smallest x-coordinate. Each interior node v of T is associated with a canonical subset $C_v \subseteq S$ containing the points stored at leaves in the subtree rooted at v. Let a_v (resp. b_v) be the smallest (resp. largest) x-coordinate of any point in C_v. The interior node v stores the values a_v and b_v and the set C_v in an array sorted by the y-coordinates of its points. The size of T is $O(n \log n)$, and it can be constructed in time $O(n \log n)$. The range-reporting query for a rectangle $q = [a_1, b_1] \times [a_2, b_2]$ can be answered by traversing T as follows. Suppose we are at a node v. If v is a leaf, then we report the point stored at v if it lies inside q. If v is an interior node and the interval $[a_v, b_v]$ does not intersect $[a_1, b_1]$, there is nothing to do. If $[a_v, b_v] \subseteq [a_1, b_1]$, we report all the points of C_v whose y-coordinates lie in the interval $[a_2, b_2]$, by performing a binary search. Otherwise, we recursively visit both children of v. The query time of this procedure is $O(\log^2 n + k)$, which can be improved to $O(\log n + k)$, using ***fractional-cascading*** (Section 30.3).

The size of the data structure can be reduced to $O(n \log n / \log\log n)$, without affecting the asymptotic query time, by constructing a range tree with $O(\log n)$ fanout and storing additional auxiliary structures at each node [21]. If the query rectangles are "3-sided rectangles" of the form $[a_1, b_1] \times [a_2, \infty]$, then one can use a ***priority search tree*** of size $O(n)$ to answer a planar range-reporting query in time $O(\log n + k)$ [63]; see [6] for a few other special cases in which the storage can be reduced to linear. All these structures can be implemented in the elementary pointer-machine model and can be dynamized using the standard partial-rebuilding technique [69]. If the preprocessing time of the data structure is $P(n)$, then a point can be inserted into or deleted from the data structure in $O((P(n)/n) \log n)$ amortized time. The update time can be made worst-case using the known deamortization techniques [36]. If we have a data structure for answering d-dimensional range-reporting queries, one can construct a $(d + 1)$-dimensional range-reporting structure in the EPM model, using multi-level range trees (see Section 31.4), by paying a $\log n$ factor in storage, preprocessing time, and query time.

If we use the RAM model, a set S of n points in $\mathbb{R}^2$ can be preprocessed into a data structure of size $O(n \log^\epsilon n)$ so that all k points lying inside a query rectangle can be reported in $O(\log n + k)$ time. Mortensen [65] has developed a data structure of size $O(n \log n / \log\log n)$ that can answer a range query in $O(\log n + k)$ time and can insert or delete a point in $O(\log n)$ time. If the points lie on a $n \times n$ grid in the plane, then a query can be answered in $O(\log\log n + k)$ time using $O(n \log^\epsilon n)$ storage or in time $O((\log\log n)^2 + k \log\log n)$ using $O(n \log\log n)$ storage. For points in $\mathbb{R}^3$, a query can be answered in $O(\log n + k)$ time using $O(n \log^{1+\epsilon} n)$ storage. As for the data structures in the pointer-machine model, the range reporting data structures in the RAM model can be extended to higher dimensions by paying a $\log n$ factor in storage and query time for each dimension. Alternatively, a d-dimensional data structure can be extended to a $(d+1)$-dimensional data structure by paying a $\log^{1+\epsilon} n$ factor in storage and a $\log n / \log\log n$ factor in the query time.

TABLE 31.2.1 Upper bounds known on orthogonal range reporting.

d	MODEL	$S(n)$	$Q(n)$
$d = 2$	RAM	n	$\log n + k\log^{\epsilon}(2n/k)$
	RAM	$n\log\log n$	$\log n + k\log\log(4n/k)$
	RAM	$n\log^{\epsilon} n$	$\log n + k$
	APM	n	$k\log(2n/k)$
	EPM	n	$k\log^2(2n/k)$
	EPM	$\frac{n\log n}{\log\log n}$	$\log n + k$
$d = 3$	RAM	$n\log^{1+\epsilon} n$	$\log n + k$
	EPM	$n\log^3 n$	$\log n + k$

The two-dimensional range tree described earlier can be used to answer a range counting query in $O(\log n)$ time using $O(n\log n)$ storage. However, if we use the RAM model in which we assume that each word stores $\log n$ bits, the size can be reduced to $O(n)$ by compressing the auxiliary information stored at each node [23].

TABLE 31.2.2 Upper bounds known on orthogonal semigroup range searching.

MODEL	$S(n)$	$Q(n)$
arithmetic	m	$\frac{n\log n}{\log(2m/n)}$
RAM	n	$\log^{2+\epsilon} n$
RAM	$n\log\log n$	$\log^2 n\log\log n$
RAM	$n\log^{\epsilon} n$	$\log^2 n$
APM	n	$\log^3 n$
EPM	n	$\log^4 n$

LOWER BOUNDS

Fredman [44, 45] was the first to prove nontrivial lower bounds on orthogonal range searching, but he considered the framework in which the points could be inserted and deleted dynamically. He showed that a mixed sequence of n insertions, deletions, and queries takes $\Omega(n\log^d n)$ time. These bounds were extended by Willard [78] to a group model, under some restrictions. Chazelle proved lower bounds for the static version of orthogonal range searching, which almost match the best upper bounds known [26]. The following theorem summarizes his main result.

THEOREM 31.2.1 *Chazelle* [26]

Let $(\mathbf{S}, \oplus)$ *be a faithful semigroup, let* d *be a constant, and let* n *and* m *be parameters. Then there exists a set* S *of* n *weighted points in* $\mathbb{R}^d$*, with weights from* $\mathbf{S}$*, such that the worst-case query time, under the semigroup model, for an orthogonal range-searching data structure that uses* m *units of storage is* $\Omega((\log n/\log(2m/n))^{d-1})$.

Theorem 31.1.1 holds even if the queries are quadrants instead of rectangles. In fact, this lower bound applies to answering the dominance query for a randomly chosen query point; in this sense the above theorem gives a lower bound on the average-case complexity of the query time. It should pointed out that Theorem 31.1.1 assumes the weights of points in S to be a part of the input. That is, the data structure is not tailored to a special set of weights, and it should work for any set of weights. It is conceivable that a faster algorithm can be developed for answering orthogonal range-counting queries, exploiting the fact that the weight of each point is 1 in this case. None of the known algorithms are able to exploit this fact, however.

A rather surprising result of Chazelle [25] shows that the size of any data structure on a pointer machine that answers a d-dimensional range-reporting query in $O(\log^c n + k)$ time, for any constant c, is $\Omega(n(\log n/\log\log n)^{d-1})$. Notice that this lower bound is greater than the known upper bound for answering two-dimensional reporting queries on the RAM model.

These lower bounds do not hold for off-line orthogonal range searching, where given a set of n weighted points in $\mathbb{R}^d$ and a set of n rectangles, one wants to compute the weight of points in each rectangle. Chazelle [28] proved that the off-line version takes $\Omega(n(\log n/\log\log n)^{d-1})$ time in the semigroup model and $\Omega(n \log\log n)$ time in the group model. For $d = \Omega(\log n)$ (resp. $d = \Omega(\log n/\log\log n)$), the lower bound for the off-line range-searching problem in the group model can be improved to $\Omega(n \log n)$ (resp. $\Omega(n \log n/\log\log n)$) [31]. The close connection between the lower bounds on range searching and the so-called discrepancy of set systems is discussed in [30].

SECONDARY MEMORY STRUCTURES

I/O-efficient orthogonal range-searching structures have received much attention recently because of massive data sets in spatial databases. The main idea underlying these structures is to construct high-degree trees instead of binary trees. For example, variants of B-trees are used to answer 1-dimensional range-searching queries [74]. Arge et al. [14] developed an external priority search tree so that a 3-sided-rectangle-reporting query can be answered in $O(\log_B \nu + \kappa)$ I/Os using $O(\nu)$ storage, where $\nu = n/B$ and $\kappa = k/B$. The main ingredient of their algorithm is a data structure that can store B^2 points using $O(B)$ blocks and can report all points lying inside a 3-sided rectangle in $O(1 + \kappa)$ I/Os. Combining their external priority search tree with Chazelle's data structure for range reporting [21], they construct an external range tree that uses $O(\nu \log_B \nu/\log\log_B \nu)$ blocks and answers a two-dimensional rectangle reporting query in time $O(\log_B n + \kappa)$. By extending the ideas proposed in [25], it cane be shown that any secondary-memory data structure that answers a range-reporting query using $O(\log_B^c \nu + \kappa)$ I/Os requires $\Omega(\nu \log_B \nu/\log\log_B n)$ storage. Govindrajan et al. [48] have shown that a two-dimensional range counting query can be answered in $O(\log_B \nu)$ I/Os using $O(\nu)$ blocks of storage, assuming that each word can store $\log n$ bits.

TABLE 31.2.3 Secondary-memory structures for orthogonal range searching. Here $\nu = n/B$, $\kappa = k/B$, and $\beta(n) = \log\log\log_B \nu$.

d	RANGE	$Q(n)$	$S(n)$
$d = 1$	interval	$\log_B \nu + \kappa$	ν
$d = 2$	3-sided rect	$\log_B \nu + \kappa$	ν
	rectangle	$\log_B \nu + \kappa$	$\nu \log_B \nu / \log\log_B \nu$
$d = 3$	octant	$\beta(\nu, B) \log_B \nu + \kappa$	$\nu \log \nu$
	box	$\beta(\nu, B) \log_B \nu + \kappa$	$\nu \log^4 \nu$

LINEAR-SIZE DATA STRUCTURES

None of the data structures described above are used in practice, even in two dimensions, because of the polylogarithmic overhead in their size. For a data structure to be used in real applications, its size should be at most cn, where c is a very small constant, the time to answer a typical query should be small — the lower bounds mentioned earlier imply that we cannot hope for small worst-case bounds — and it should support insertions and deletions of points. Keeping these goals in mind, a plethora of data structures have been proposed.

The most widely used data structures for answering 1-dimensional range queries are B-trees and their variants. Since a B-tree requires a linear order on the input elements, several techniques such as Lexicographic ordering, bit interleaving, and space filling curves have been used define a linear ordering on points in higher dimensions in order to store then in a B-tree. A more efficient approach to answer high-dimensional range queries is to construct a recursive partition of space, typically into rectangles, and to construct a tree induced by this partition. The simplest example of this type of data structure is the ***quad tree*** in the plane. A quad tree is a 4-way tree, each of whose nodes is associated with a square R_v. R_v is partitioned into four equal-size squares, each of which is associated with one of the children of v. The squares are partitioned until at most one point is left inside a square. A range-search query can be answered by traversing the quad tree in a top-down fashion. Because of their simplicity, quad trees are one of the most widely used data structures for a variety of problems. One disadvantage of quad trees is that arbitrarily many levels of partitioning may be required to separate tightly clustered points. Finkel and Bentley [42] described a variant of the quad tree for range searching, called a *point quad-tree*, in which each node is associated with a rectangle and the rectangle is partitioned into four rectangles by choosing a point in the interior and drawing horizontal and vertical lines through that point. Typically the point is chosen so that the height of the tree is $O(\log n)$. In order to minimize the number of disk accesses, one can partition the square into many squares (instead of four) by a drawing either a uniform or a nonuniform grid. The ***grid file*** data structure, introduced by Nievergelt et al. [67], is based on this idea.

Quad trees and their variants construct a grid on a rectangle containing all the input points. One can instead partition the enclosing rectangle into two rectangles by drawing a horizontal or a vertical line and partitioning each of the two rectangles independently. This is the idea behind the ***kd-tree*** data structure by Bentley [16]. In particular, a *kd*-tree is a binary tree, each of whose nodes v is associated with a rectangle R_v. If R_v does not contain any point in its interior, v is a leaf. Otherwise, R_v is partitioned into two rectangles by drawing a horizontal or vertical line so that each rectangle contains at most half of the points; splitting lines are alternately horizontal and vertical. In order to minimize the number of disk accesses, Robinson [72] generalized a *kd*-tree to a ***kd-B-tree***, in which one constructs a B-tree instead of a binary tree on the recursive partition of the enclosing rectangle, so all leaves of the tree are at the same level and each node has between $B/2$ and B children. The rectangles associated with the children are obtained by splitting R_v recursively, as in a *kd*-tree. A simple top-down approach to construct a *kd*-B-tree requires $O(\nu \log_2 \nu)$ I/Os, but the preprocessing cost can be reduced to $O(\nu \log_B \nu)$ I/Os using a more sophisticated approach [3].

If points are dynamically inserted into a *kd*-tree or *kd*-B-tree, then some of the nodes may have to be split, which is an expensive operation because splitting a node may require reconstructing the entire subtree rooted at that node. A few variants of *kd*-trees have been proposed that can update the structure in $O(\text{polylog} n)$ time and can answer a query in $O(\sqrt{n}+k)$ time. On the practical side, many variants of *kd*-B-trees have also been proposed to minimize the number of splits, to optimize the space, and to improve the query time, most notable of them are ***buddy trees*** [75] and ***hB-trees*** [58, 41]. A buddy tree is a combination of quad- and *kd*-B-trees in the sense that rectangles are split into sub-rectangles only at some specific locations, which simplifies the split procedure. If points are in degenerate position, then it may not be possible to split a square into two halves by a line. Lomen and Salzberg [58] circumvent this problem by introducing a new data structure, called *hB-tree*, in which the region associated with a node is allowed to be $R_1 \setminus R_2$ where R_1 and R_2 are rectangles. A more refined version of this data structure, known as *hB*$^{\Pi}$*-tree*, is presented in [41].

All the data structures described in this section for d-dimensional range searching construct a recursive partition of $\mathbb{R}^d$. There are other data structures that construct a hierarchical cover of $\mathbb{R}^d$, most popular of which is the ***R-tree***, originally introduced by Guttman [52]. An R-tree is a B-tree, each of whose nodes stores a set of rectangles. Each leaf stores a subset of input point, and each input point is stored at exactly one leaf. For each node v, let R_v be the smallest rectangle containing all the rectangles stored at v; R_v is stored at the parent of v (along with the pointer to v). R_v induces the subspace corresponding to the subtree rooted at v, in the sense that for any query rectangle intersecting R_v, the subtree rooted at v is searched. Rectangles stored at a node are allowed to overlap. Although allowing rectangles to overlap helps reduce the size of the data structure, answering a query becomes more expensive. Guttman suggests a few heuristics to construct a R-tree so that the overlap is minimized. Several better heuristics for improving the performance minimizing the overlap have been proposed, including R*- and Hilbert-R-trees. Actually, an R-tree can be constructed on a set of rectangles. Agarwal et al. [5] showed how to construct an R-tree on a set of n rectangles in $\mathbb{R}^d$ so that all k rectangles intersecting a query rectangle can be reported in $O(n^{1-1/d}+k)$ time.

PARTIAL-SUM QUERIES

Preprocess a d-dimensional array A with n entries, in an additive semigroup, into a data structure, so that for a d-dimensional rectangle $\gamma = [a_1, b_1] \times \ldots \times [a_d, b_d]$, the sum

$$\sigma(A, \gamma) = \sum_{(k_1, k_2, \ldots, k_d) \in \gamma} A[k_1, k_2, \ldots, k_d]$$

can be computed efficiently. In the offline version, given A and m rectangles $\gamma_1, \gamma_2, \ldots, \gamma_m$, we wish to compute $\sigma(A, \gamma_i)$ for each i. Note that this is just a special case of orthogonal range searching, where the points lie on a regular d-dimensional lattice.

Partial-sum queries are widely used for on-line analytical processing (OLAP) of commercial databases. OLAP allows companies to analyze aggregate databases built from their data warehouses. A popular data model for OLAP applications is the multidimensional database, known as ***data cube*** [49], which represents the data as d-dimensional array. Thus, an aggregate query can be formulated as a partial-sum query. Driven by this application, several heuristics have been proposed to answer partial-sum queries on data cubes [56, 55] and the references therein.

Yao [80] showed that, for $d = 1$, a partial-sum query can be answered in $O(\alpha(n))$ time using $O(n)$ space, where $\alpha(n)$ is the inverse Ackermann function. If the additive operator is *max* or *min*, then a partial-sum query can be answered in $O(1)$ time under the RAM model using a Cartesian tree, developed by Vuillemin [77].

For $d > 1$, Chazelle and Rosenberg [32] developed a data structure of size $O(n \log^{d-1} n)$ that can answer a partial-sum query in time $O(\alpha(n) \log^{d-2} n)$. They also showed that the offline version that answers m given partial-sum queries on n points takes $\Omega(n + m\alpha(m, n))$ time for any fixed $d \geq 1$. If points are allowed to insert into S, the query time is $\Omega(\log n / \log \log n)$ [43, 81] for the one-dimensional case; the bounds were extended by Chazelle [26] to $\Omega((\log n / \log \log n)^d)$, for any fixed dimension d.

31.3 SIMPLEX RANGE SEARCHING

Unlike orthogonal range searching, no simplex range-searching data structure is known that can answer a query in polylogarithmic time using near-linear storage. In fact, the lower bounds stated below indicate that there is little hope of obtaining such a data structure, since the query time of a linear-size data structure, under the semigroup model, is roughly at least $n^{1-1/d}$ (thus only saving a factor of $n^{1/d}$ over the naive approach). Because the size and query time of any data structure have to be at least linear and logarithmic, respectively, we consider these two ends of the spectrum: (i) how fast a simplex range query can be answered using a linear-size data structure; and (ii) how large the size of a data structure should be in order to answer a query in logarithmic time. Combining these two extreme cases leads to a space/query-time tradeoff.

GLOSSARY

Arrangements: The arrangement of a set H of hyperplanes in $\mathbb{R}^d$ is the subdivision of $\mathbb{R}^d$ into cells of dimension k, for $0 \le k \le d$, each cell of dimension $k < d$ being a maximal connected set contained in the intersection of a fixed subset of H and not intersecting any other hyperplane of H. See Chapter 21.

$1/r$-cutting: Let H be a set of n hyperplanes in $\mathbb{R}^d$ and let $1 \le r \le n$ be a parameter. A $(1/r)$-cutting of H is a set of (relatively open) disjoint simplices covering $\mathbb{R}^d$ so that each simplex intersects at most n/r hyperplanes of H.

Duality: The dual of a point $(a_1, \ldots, a_d) \in \mathbb{R}^d$ is the hyperplane $x_d = -a_1x_1 - \cdots - a_{d-1}x_{d-1} + a_d$, and the dual of a hyperplane $x_d = b_1x_1 + \cdots + b_d$ is the point $(b_1, \ldots, b_{d-1}, b_d)$.

LINEAR-SIZE DATA STRUCTURES

Most of the linear-size data structures for simplex range searching are based on ***partition trees***, originally introduced by Willard [79] for a set of points in the plane. Roughly speaking, a partition tree is a hierarchical decomposition scheme (in the sense described in Section 31.1) that recursively partitions the points into canonical subsets and encloses each canonical subset by a simple convex region (e.g. simplex), so that any hyperplane intersects only a fraction of the regions associated with the "children" of a canonical subset. A query is answered as described in Section 31.1. The query time depends on the maximum number of children regions of a node that a hyperplane can intersect. The partition tree proposed by Willard partitions each canonical subsets into four children, each contained in a wedge so that any line intersects at most three of them. As a result, the time spent in reporting all k points lying inside a triangle is $O(n^{0.792} + k)$. A number of partition trees with improved query time were introduced later, but a major breakthrough in simplex range searching was made by Haussler and Welzl [53]. They formulated range searching in an abstract setting and, using elegant probabilistic methods, gave a randomized algorithm to construct a linear-size partition tree with $O(n^\alpha)$ query time, where $\alpha = 1 - \frac{1}{d(d-1)+1} + \epsilon$ for any $\epsilon > 0$. The best-known linear-size data structure for simplex range searching, which almost matches the lower bounds mentioned below, is by Matoušek [61]. He showed that a simplex range-counting (resp. range-reporting) query in $\mathbb{R}^d$ can be answered in time $O(n^{1-1/d})$ (resp. $O(n^{1-1/d} + k)$). His algorithm is based on a stronger version of the following theorem.

THEOREM 31.3.1 *Matoušek* [60]

Let S be a set of n points in $\mathbb{R}^d$, and let $1 < r \le n/2$ be a given parameter. Then there exists a family of pairs $\Pi = \{(S_1, \Delta_1), \ldots, (S_m, \Delta_m)\}$ such that $S_i \subseteq S$ lies inside simplex Δ_i, $n/r \le |S_i| \le 2n/r$, $S_i \cap S_j = \emptyset$ for $i \ne j$, and every hyperplane crosses at most $cr^{1-1/d}$ simplices of Π; here c is a constant. If r is a constant, then Π can be constructed in $O(n)$ time.

Using this theorem, a partition tree T can be constructed as follows. Each interior node v of T is associated with a subset $S_v \subseteq S$ and a simplex Δ_v containing S_v; the root of T is associated with S and $\mathbb{R}^d$. Choose r to be a sufficiently large constant. If $|S| \le 4r$, T consists of a single node, and it stores all points of S.

Otherwise, we construct a family of pairs $\Pi = \{(S_1, \Delta_1), \ldots, (S_m, \Delta_m)\}$ using Theorem 31.3.1. We recursively construct a partition tree T_i for each S_i and attach T_i as the ith subtree of u. The root of T_i also stores Δ_i. The total size of the data structure is linear, and it can be constructed in time $O(n \log n)$. Since any hyperplane intersects at most $cr^{1-1/d}$ simplices of Π, the query time of simplex range reporting is $O(n^{1-1/d} \cdot n^{\log_r c} + k)$; the $\log_r c$ factor can be reduced to any arbitrarily small positive constant ϵ by choosing r sufficiently large. Although the query time can be improved to $O(n^{1-1/d} \log^c n + k)$ by choosing r to be n^ϵ, a stronger version of Theorem 31.3.1, which was proved in [61], and some other sophisticated techniques are needed to obtain $O(n^{1-1/d} + k)$ query time.

If the points in S lie on a b-dimensional algebraic surface of constant degree, a simplex range-counting query can be answered in time $O(n^{1-\gamma})$ using linear space, where $\gamma = 1/\lfloor (d+b)/2 \rfloor$. We note that better bounds can be obtained for halfspace range reporting, using ***filtering search***; see Table 31.3.1. A halfspace range-reporting query in the I/O model can be answered in $O(\log_B \nu + \kappa)$ I/Os using $O(\nu)$ (resp. $O(\nu \log_B \nu)$) blocks of storage for $d = 2$ (resp. $d = 3$) [2].

TABLE 31.3.1 Near-linear-size data structures for halfspace range searching.

d	$S(n)$	$Q(n)$	NOTES
$d = 2$	n	$\log n + k$	reporting
	n	$\log n$	emptiness
$d = 3$	$n \log\log n$	$\log n + k$	reporting
	n	$\log^2 n + k$	reporting
	n	$\log n$	emptiness
$d > 3$	$n \log\log n$	$n^{1-1/\lfloor d/2 \rfloor} \log^c n + k$	reporting
	n	$n^{1-1/d} 2^{O(\log^* n)}$	emptiness
even d	n	$n^{1-1/\lfloor d/2 \rfloor} \log^c n + k$	reporting

DATA STRUCTURES WITH LOGARITHMIC QUERY TIME

For the sake of simplicity, we first consider the halfspace range-counting problem. Using a standard duality transform, this problem can be reduced to the following: *Given a set H of n hyperplanes, determine the number of hyperplanes of H lying above a query point.* Since the same subset of hyperplanes lies above all points in a single cell of $\mathcal{A}(\mathcal{H})$, the arrangement of H, we can answer a halfspace range-counting query by locating the cell of $\mathcal{A}(\mathcal{H})$ that contains the point dual to the hyperplane bounding the query halfspace. The following theorem of Chazelle [27] yields an $O((n/\log n)^d)$-size data structure, with $O(\log n)$ query time, for halfspace range counting.

THEOREM 31.3.2 *Chazelle* [27]

Let H be a set of n hyperplanes and $r \le n$ a parameter. Set $s = \lceil \log_2 r \rceil$. There exist k cuttings $\Xi_1, \ldots, \Xi_s$ so that Ξ_i is a $(1/2^i)$-cutting of size $O(2^{id})$, each simplex of Ξ_i is contained in a simplex of Ξ_{i-1}, and each simplex of Ξ_{i-1} contains a constant number of simplices of Ξ_i. Moreover, $\Xi_1, \ldots, \Xi_s$ can be computed in time $O(nr^{d-1})$.

The above approach can be extended to the simplex range-counting problem as well. That is, store the solution of every combinatorially distinct simplex (two simplices are combinatorially distinct if they do not contain the same subset of S). Since there are $\Theta(n^{d(d+1)})$ combinatorially distinct simplices, such an approach will require $\Omega(n^{d(d+1)})$ storage. Chazelle et al. [34] showed that the size can be reduced to $O(n^{d+\epsilon})$, for any $\epsilon > 0$, using a multi-level data structure, with each level composed of a halfspace range-counting data structure. The space bound can be reduced to $O(n^d)$ by increasing the query time to $O(\log^{d+1} n)$ [61]. Halfspace range-reporting queries can be answered in $O(\log n + k)$ time, using $O(n^{\lfloor d/2 \rfloor} \text{polylog} n)$ space.

A space/query-time tradeoff for simplex range searching can be attained by combining the linear-size and logarithmic query-time data structures. The known results on this tradeoff are summarized in Table 31.3.2. $Q(m,n)$ is teh query time on n points using m units of storage.

TABLE 31.3.2 Space/query-time tradeoff.

RANGE	MODE	$Q(m,n)$
Simplex	reporting	$\frac{n}{m^{1/d}} \log^{d+1} \frac{m}{n} + k$
Simplex	counting	$\frac{n}{m^{1/d}} \log^{d+1} \frac{m}{n}$
Halfspace	reporting	$\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log^c n + k$
Halfspace	emptiness	$\frac{n}{m^{1/\lfloor d/2 \rfloor}} \log^c n$
Halfspace	counting	$\frac{n}{m^{1/d}} \log \frac{m}{n}$

LOWER BOUNDS

Fredman [46] showed that a sequence of n insertions, deletions, and halfplane queries on a set of points in the plane requires $\Omega(n^{4/3})$ time, in the semigroup model. His technique, however, does not extend to static data structures. In a series of papers, Chazelle has proved nontrivial lower bounds on the complexity of online simplex range searching, using various elegant mathematical techniques. The following theorem is perhaps the most interesting result on lower bounds.

THEOREM 31.3.3 *Chazelle* [24]

Let $(\mathbf{S}, \oplus)$ be a faithful semigroup, let n, m be positive integers such that $n \le m \le n^d$, and let S be a random set of weighted points in $[0,1]^d$ with weights from $\mathbf{S}$. If *only m words of storage is available, then with high probability, the worst-case query time for a simplex range query in S is $\Omega(n/\sqrt{m})$ for $d = 2$, or $\Omega(n/(m^{1/d}\log n))$ for $d \geq 3$, in the semigroup model.*

It should be pointed out that this theorem holds even if the query ranges are wedges or strips, but it does not hold if the ranges are hyperplanes. Chazelle and Rosenberg [33] proved a lower bound of $\Omega(n^{1-\epsilon}/m + k)$ for simplex range reporting under the pointer-machine model. These lower bounds do not hold for halfspace range searching. A somewhat weaker lower bound for halfspace queries was proved by Brönnimann et al. [19].

As we saw earlier, faster data structures are known for halfspace emptiness queries. A series of papers by Erickson established the first nontrivial lower bounds for online and offline emptiness query problems, in the partition-graph model of computation. He first considered this model for Hopcroft's problem — Given a set of n points and m lines, does any point lie on a line? — for which he obtained a lower bound of $\Omega(n \log m + n^{2/3}m^{2/3} + m \log n)$ [39], almost matching the best known upper bound $O(n \log m + n^{2/3}m^{2/3}2^{O(\log^*(n+m))} + m \log n)$, due to Matoušek [61]. He later established lower bounds on a tradeoff between space and query time, or preprocessing and query time, for online hyperplane emptiness queries [40]. For d-dimensional hyperplane queries, $\Omega(n^d/\text{polylog}n)$ preprocessing time is required to achieve polylogarithmic query time, and the best possible query time is $\Omega(n^{1/d}/\text{polylog}n)$ if only $O(n\text{polylog}n)$ preprocessing time is allowed. More generally, in two dimensions, if the preprocessing time is p, the query time is $\Omega(n/\sqrt{p})$.

Table 31.3.3 summarizes the best known lower bounds for online simplex queries. Lower bounds for emptiness problems apply to counting and reporting problems as well.

TABLE 31.3.3 Lower bounds for online simplex range searching using $O(m)$ space.

Range	Problem	Model	Query Time
Simplex	Semigroup	Semigroup ($d = 2$)	$n/\sqrt{m}$
	Semigroup	Semigroup ($d > 2$)	$n/(m^{1/d}\log n)$
	Reporting	Pointer machine	$n^{1-\epsilon}/m^{1/d} + k$
Hyperplane	Semigroup	Semigroup	$(n/m^{1/d})^{2/(d+1)}$
	Emptiness	Partition graph	$(n/\log n)^{\frac{d^2+1}{d^2+d}} \cdot (1/m^{1/d})$
Halfspace	Semigroup	Semigroup	$(n/\log n)^{\frac{d^2+1}{d^2+d}} \cdot (1/m^{1/d})$
	Emptiness	Partition graph	$(n/\log n)^{\frac{\delta^2+1}{\delta^2+\delta}} \cdot (1/m^{1/\delta})$, where $d \geq \delta(\delta+3)/2$

OPEN PROBLEMS

1. Bridge the gap between the known upper and lower bounds in the group model. Even in the semigroup model there is a small gap between the known bounds.

2. Can a halfspace range-reporting query be answered in $O(n^{1-1/\lfloor d/2 \rfloor} + k)$ time using linear space if d is odd?

31.4 VARIANTS AND EXTENSIONS

In this section we review a few extensions of range-searching data structures: multi-level structures, semialgebraic range searching, and kinetic range searching.

GLOSSARY

Semialgebraic set: A subset of $\mathbb{R}^d$ obtained as a finite Boolean combination of sets of the form $\{f \geq 0\}$, where f is a d-variate polynomial (see Chapter 29).

Tarski cells: A simply connected real semialgebraic set defined by a constant number of polynomials, each of constant degree.

MULTI-LEVEL STRUCTURES

A rather powerful property of data structures based on decomposition schemes (described in Section 31.1) is that they can be cascaded together to answer more complex queries, at the increase of a logarithmic factor per level in their performance. The real power of the cascading property was first observed by Dobkin and Edelsbrunner [37], who used this property to answer several complex geometric queries. Since their result, several papers have exploited and extended this property to solve numerous geometric-searching problems. We briefly sketch the general cascading scheme.

Let S be a set of weighted objects. Recall that a geometric-searching problem $\mathcal{P}$, with underlying relation $\Diamond$, requires computing $\sum_{p \Diamond \gamma} w(p)$ for a query range γ. Let $\mathcal{P}^1$ and $\mathcal{P}^2$ be two geometric-searching problems, and let $\Diamond^1$ and $\Diamond^2$ be the corresponding relations. Then we define $\mathcal{P}^1 \circ \mathcal{P}^2$ to be the conjunction of $\mathcal{P}^1$ and $\mathcal{P}^2$, whose relation is $\Diamond^1 \cap \Diamond^2$. That is, for a query range γ, we want to compute $\sum_{p \Diamond^1 \gamma, p \Diamond^2 \gamma} w(p)$. Suppose we have hierarchical decomposition schemes $\mathcal{D}^1$ and $\mathcal{D}^2$ for problems $\mathcal{P}^1$ and $\mathcal{P}^2$. Let $\mathcal{F}^1 = \mathcal{F}^1(S)$ be the set of canonical subsets constructed by $\mathcal{D}^1$, and for a range γ, let $\mathcal{C}^1_\gamma = \mathcal{C}^1(S, \gamma)$ be the corresponding partition of $\{p \in S \mid p \Diamond^1 \gamma\}$ into canonical subsets. For each canonical subset $C \in \mathcal{F}^1$, let $\mathcal{F}^2(C)$ be the collection of canonical subsets of C constructed by $\mathcal{D}^2$, and let $\mathcal{C}^2(C, \gamma)$ be the corresponding partition of $\{p \in C \mid p \Diamond^2 \gamma\}$ into level-two canonical subsets. The decomposition scheme $\mathcal{D}^1 \circ \mathcal{D}^2$ for the problem $\mathcal{P}^1 \circ \mathcal{P}^2$ consists of the canonical subsets $\mathcal{F} = \bigcup_{C \in \mathcal{F}^1} \mathcal{F}^2(C)$. For a query range γ, the query output is $\mathcal{C}_\gamma = \bigcup_{C \in \mathcal{C}^1_\gamma} \mathcal{C}^2(C, \gamma)$. Note that we can cascade any number of decomposition schemes in this manner.

If we view $\mathcal{D}^1$ and $\mathcal{D}^2$ as tree data structures, then cascading the two decomposition schemes can be regarded as constructing a two-level tree, as follows. We first construct the tree induced by $\mathcal{D}^1$ on S. Each node v of $\mathcal{D}^1$ is associated with a canonical subset C_v. We construct a second-level tree $\mathcal{D}^2_v$ on C_v and store $\mathcal{D}^2_v$ at v as its secondary structure. A query is answered by first identifying the nodes that correspond to the canonical subsets $C_v \in \mathcal{C}^1_\gamma$ and then searching the corresponding secondary trees to compute the second-level canonical subsets $\mathcal{C}^2(C_v, \gamma)$.

Suppose the size and query time of each decomposition scheme are at most $S(n)$ and $Q(n)$, respectively, and $\mathcal{D}^1$ is efficient and r-convergent (cf. Section 31.1), for some constant $r > 1$, then the size and query time of the decomposition scheme $\mathcal{D}$ are $O(S(n)\log_r n)$ and $O(Q(n)\log_r n)$, respectively. If $\mathcal{D}^2$ is also efficient and r-convergent, then $\mathcal{D}$ is also efficient and r-convergent. In some cases, the logarithmic overhead in the query time or the space can be avoided.

The real power of multi-level data structures stems from the fact that there are no restrictions on the relations $\Diamond^1$ and $\Diamond^2$. Hence, any query that can be represented as a conjunction of a constant number of "primitive" queries, each of which admits an efficient, r-convergent decomposition scheme, can be answered by cascading individual decomposition schemes. We will describe a few multi-level data structures in this and the following sections.

SEMIALGEBRAIC RANGE SEARCHING

So far we have assumed that the ranges were bounded by hyperplanes, but in many applications one has to deal with ranges bounded by nonlinear functions. For example, a query of the form, "for a given point p and a real number r, find all points of S lying within distance r from p," is a range-searching problem in which the ranges are balls.

As shown below, ball range searching in $\mathbb{R}^d$ can be formulated as an instance of the halfspace range searching in $\mathbb{R}^{d+1}$. So a ball range-reporting (resp. range-counting) query in $\mathbb{R}^d$ can be answered in time $O(n/m^{1/\lceil d/2 \rceil}\log^c n + k)$ (resp. $O(n/m^{1/(d+1)}\log(m/n)))$, using $O(m)$ space; somewhat better performance can be obtained using a more direct approach (Table 31.4.1). However, relatively little is known about range-searching data structures for more general ranges.

A natural class of nonlinear ranges is the family of Tarski cells. It suffices to consider the ranges bounded by a single polynomial because the ranges bounded by multiple polynomials can be handled using multi-level data structures. We assume that the ranges are of the form

$$\Gamma_f(a) = \{x \in \mathbb{R}^d \mid f(x,a) \geq 0\},$$

where f is a $(d{+}p)$-variate polynomial specifying the type of ranges (disks, cylinders, cones, etc.), and a is a p-tuple specifying a specific range of the given type (e.g., a specific disk). We will refer to the range-searching problem in which the ranges are from the set Γ_f as ***Γ_f-range searching.***

One approach to answering Γ_f-range queries is ***linearization***. We represent the polynomial $f(x,a)$ in the form

$$f(x,a) = \psi_0(a) + \psi_1(a)\varphi_1(x) + \cdots + \psi_\lambda(a)\varphi_\lambda(x)$$

where $\varphi_1, \ldots, \varphi_\lambda, \psi_0, \ldots, \psi_\lambda$ are real functions. A point $x \in \mathbb{R}^d$ is mapped to the point

$$\varphi(x) = (\varphi_1(x), \varphi_2(x), \ldots, \varphi_\lambda(x)) \in \mathbb{R}^\lambda.$$

Then a range $\gamma_f(a) = \{x \in \mathbb{R}^d \mid f(x,a) \geq 0\}$ is mapped to a halfspace

$$\varphi^\#(a) : \{y \in \mathbb{R}^\lambda \mid \psi_0(a) + \psi_1(a)y_1 + \cdots + \psi_\lambda(a)y_\lambda \geq 0\};$$

TABLE 31.4.1 Semialgebraic range counting; λ is the dimension of linearization.

d	RANGE	$S(n)$	$Q(n)$	NOTES
$d = 2$	disk	$n \log n$	$\sqrt{n \log n}$	
$d \leq 4$	Tarski cell	n	$n^{1-1/d+\epsilon}$	partition tree
$d \geq 4$	Tarski cell	n	$n^{1-\frac{1}{2d-4}+\epsilon}$	partition tree
	Tarski cell	n	$n^{1-\frac{1}{\lambda}+\epsilon}$	linearization
	disk	n	$n^{1-\frac{1}{d}+\epsilon}$	linearization

λ is called the ***dimension*** of linearization. For example, a set of spheres in $\mathbb{R}^d$ admit a linearization of dimension $d+1$, using the well-known lifting transform. Agarwal and Matoušek [10] have described an algorithm for computing a linearization of the smallest dimension under certain assumptions on φ_i's and ψ_i's. If f admits a linearization of dimension λ, a Γ_f-range query can be answered using a λ-dimensional halfspace range-searching data structure. Agarwal and Matoušek [10] have also proposed another approach to answer Γ_f-range queries, by extending Theorem 31.3.1 to Tarski cells and by constructing partition trees using this extension. Table 31.4.1 summarizes the known results on Γ_f-range-counting queries. The bounds mentioned in the third row of the table rely on the recent result by Koltun [57] on the vertical decomposition of arrangements of surfaces.

KINETIC RANGE SEARCHING

Let $S = \{p_1, \ldots, p_n\}$ be a set of n points in $\mathbb{R}^2$, each moving continuously. Let $p_i(t)$ denote the position of p_i at time t, and let $S(t) = \{p_1(t), \ldots, p_n(t)\}$. We assume that each point p_i is moving with fixed velocity, i.e., $p_i(t) = a_i + b_i t$ for $a_i, b_i \in \mathbb{R}^2$, and the trajectory of a point p_i is a line $\overline{p}_i$. Let L denote the set of lines corresponding to the trajectories of points in S.

We consider the following two range-reporting queries:

Q1. Given an axis-aligned rectangle R in the xy-plane and a time value t_q, report all points of S that lie inside R at time t_q, i.e., report $S(t_q) \cap R$; t_q is called the *time stamp* of the query.

Q2. Given a rectangle R and two time values $t_1 \leq t_2$, report all points of S that lie inside R at any time between t_1 and t_2, i.e., report $\bigcup_{t=t_1}^{t_2} (S(t) \cap R)$.

Two general approaches have been proposed to preprocess moving points for range searching. The first approach, which is known as the *time-oblivious* approach, regards time as a new dimension and stores the trajectories $\bar{p}_i$ of input points p_i. One can either preprocess the trajectories themselves using various techniques, or one can work in a parametric space, map each trajectory to a point in this space, and build a data structure on these points. An advantage of the time-oblivious scheme is that the data structure is updated only if the trajectory of a point changes or if a point is inserted into or deleted from the index. Since this approach preprocesses either curves in $\mathbb{R}^2$ or points in higher dimensions, the query time tends to be large. For example, if S is a set of points moving in $\mathbb{R}^1$, then the trajectory of each point is a line in $\mathbb{R}^2$ and a Q1 query corresponds to reporting all lines of L that intersect a query segment σ parallel to the x-axis. As we will see below, L can be preprocessed into a data structure of linear size so that all lines intersecting σ can be reported in $O(n^{1/2+\epsilon} + k)$ time. A similar structure can answer Q2 queries within the same asymptotic time bound. The lower bounds on simplex range searching suggest that one cannot hope to answer a query in $O(\log n + k)$ time using this approach. If S is a set of points moving in $\mathbb{R}^2$, then a Q1 query asks for reporting all lines of L that intersect a query rectangle R parallel to the xy-plane (in the xyt-space). Note that a line ℓ in $\mathbb{R}^3$ (xyt-space) intersects R if and only if their projections onto the xt- and yt-planes both intersect. Using this observation one can construct a two-level partition tree of size $O(n)$ to report in $O(n^{1/2+\epsilon} + k)$ time all lines of L intersecting R [1]. Again a Q2 query can be answered within the same time bound.

The second approach, based on the *kinetic-data-structure* framework [50], builds a dynamic data structure on the moving points. Roughly speaking, at any time it maintains a data structure on the current configuration of the points. As the points move, the data structure evolves. The main observation is that although the points are moving continuously, the data structure is updated only at discrete time instances when certain *events* occur, e.g., when any of the coordinates of two points become equal. This approach leads to fast query time, but it comes at the cost of updating the structure periodically even if the trajectory of no point changes. Another disadvantage of this approach is that it can answer a query only at the current configurations of points, though it can be extended to handle queries arriving in chronological order, i.e., the time stamps of queries are in nondecreasing order. In particular, if S is a set of points moving in $\mathbb{R}^1$, using a kinetic B-tree, a one-dimensional Q1 query can be answered in $O(\log n + k)$ time. The data structure processes $O(n^2)$ events, each of which requires $O(\log n)$ time. Similarly, by kinetizing range trees, a two-dimensional Q1 query can be answered in $O(\log n + k)$ time; the data structure processes $O(n^2)$ events, each of which requires $O(\log^2 n/\log\log n)$ time [1].

Since range trees are too complicated, a more practical approach is to use the kinetic-data-structure framework on *kd*-trees, as proposed by Agarwal et al. [7]. They propose two variants of of kinetic *kd*-trees, each of which answers Q1 queries that arrive in chronological order in $O(n^{1/2+\epsilon})$ time, for any constant $\epsilon > 0$, process $O(n^2)$ kinetic events, and spend $O(\text{polylog} n)$ time at each event. Since kinetic *kd*-trees process too many events because of the strong invariants they maintain, kinetic R-trees have also been proposed [76, 70], which typically require weaker invariants and thus are updated less frequently.

OPEN PROBLEMS

1. Can a ball range-counting query be answered in $O(\log n)$ time using $O(n^2)$ space?

2. If the hyperplanes bounding the query halfspaces satisfy some property—e.g., all of them are tangent to a given sphere—can a halfspace range-counting query be answered more efficiently?

3. Is there a simple, linear-size kinetic data structure that can answer Q1 queries in $O(\sqrt{n} + k)$ time and processes near-linear events, each requiring $O(\log^c n)$

time.

31.5 INTERSECTION SEARCHING

A general intersection-searching problem can be formulated as follows: *Given a set S of objects in $\mathbb{R}^d$, a semigroup $(\mathbf{S}, +)$, and a weight function $w : S \to \mathbf{S}$; we wish to preprocess S into a data structure so that for a query object γ, we can compute the weighted sum $\sum w(p)$, where the sum is taken over all objects of S that intersect γ.* Range searching is a special case of intersection-searching in which S is a set of points.

An intersection-searching problem can be formulated as a semialgebraic range-searching problem by mapping each object $p \in S$ to a point $\varphi(p)$ in a parametric space $\mathbb{R}^\lambda$ and every query range γ to a semialgebraic set $\psi(\gamma)$ so that p intersects γ if and only if $\varphi(p) \in \psi(\gamma)$. For example, let both S and the query ranges be sets of segments in the plane. Each segment $e \in S$ with left and right endpoints (p_x, p_y) and (q_x, q_y), respectively, can be mapped to a point $\varphi(e) = (p_x, p_y, q_x, q_y)$ in $\mathbb{R}^4$, and a query segment γ can be mapped to a semialgebraic region $\psi(\gamma)$ so that γ intersects e if and only if $\psi(\gamma) \in \varphi(e)$. A shortcoming of this approach is that λ, the dimension of the parametric space, is typically much larger than d, and thereby affecting the query time aversely. The efficiency can be significantly improved by expressing the intersection test as a conjunction of simple primitive tests (in low dimensions) and using a multi-level data structure to perform these tests. For example, a segment γ intersects another segment e if the endpoints of e lie on the opposite sides of the line containing γ and vice-versa. We can construct a two-level data structure—the first level sifts the subset $S_1 \subseteq S$ of all the segments that intersect the line supporting the query segment, and the second level reports those segments of S_1 whose supporting lines separate the endpoints of γ. Each level of this structure can be implemented using a two-dimensional simplex range-searching searching structure, and hence a reporting query can be answered in $O(n/\sqrt{m} \log^c n + k)$ time using $O(m)$ space.

It is beyond the scope of this chapter to cover all intersection-searching problems. Instead, we discuss a selection of basic problems that have been studied extensively. All intersection-counting data structures described here can answer intersection-reporting queries at an additional cost proportional to the output size. In some cases an intersection-reporting query can be answered faster. Moreover, using intersection-reporting data structures, intersection-detection queries can be answered in time proportional to their query-search time. Finally, all the data structures described in this section can be dynamized at an expense of an $O(n^\epsilon)$ factor in the storage and query time.

POINT INTERSECTION SEARCHING

Preprocess a set S of objects (e.g., balls, halfspaces, simplices, Tarski cells) in $\mathbb{R}^d$ into a data structure so that the objects of S containing a query point can be reported (or counted) efficiently. This is the inverse of the range-searching problem, and it can also be viewed as locating a point in the subdivision induced by the objects in

S. Table 31.5.1 gives some of the known results.

TABLE 31.5.1 Point intersection searching.

d	OBJECTS	$S(n)$	$Q(n)$	NOTES
$d = 2$	disks	m	$(n/\sqrt{m})^{4/3}$	counting
	disks	$n \log n$	$\log n + k$	reporting
	triangles	m	$\frac{n}{\sqrt{m}} \log^3 n$	counting
	fat triangles	$n \log^2 n$	$\log^3 n + k$	reporting
	Tarski cells	$n^{2+\epsilon}$	$\log n$	counting
$d = 3$	functions	$n^{1+\epsilon}$	$\log n + k$	reporting
	Tarski cells	$n^{3+\epsilon}$	$\log n$	counting
$d \geq 3$	simplices	m	$\frac{n}{m^{1/d}} \log^{d+1} n$	counting
	balls	$n^{d+\epsilon}$	$\log n$	counting
	balls	m	$\frac{n}{m^{1/\lceil d/2 \rceil}} \log^c n + k$	reporting
$d \geq 4$	Tarski cells	$n^{2d-4+\epsilon}$	$\log n$	counting

SEGMENT INTERSECTION SEARCHING

Preprocess a set of objects in $\mathbb{R}^d$ into a data structure so that the objects of S intersected by a query segment can be reported (or counted) efficiently. See Table 31.5.2 for some of the known results on segment intersection searching. For the sake of clarity, we have omitted polylogarithmic factors from the query-search time whenever it is of the form n/m^α.

TABLE 31.5.2 Segment intersection searching.

d	OBJECTS	$S(n)$	$Q(n)$	NOTES
$d = 2$	simple polygon	n	$(k + 1) \log n$	reporting
	segments	m	$n/\sqrt{m}$	counting
	circles	$n^{2+\epsilon}$	$\log n$	counting
	circular arcs	m	$n/m^{1/3}$	counting
$d = 3$	planes	m	$n/m^{1/3}$	counting
	spheres	m	$n/m^{1/3}$	counting
	triangles	m	$n/m^{1/4}$	counting

A special case of segment intersection searching, in which the objects are horizontal segments in the plane and query ranges are vertical segments, has been widely studied. In this case a query can be answered in time $O(\log n + k)$ using $O(n \log \log n)$ space and $O(n \log n)$ preprocessing (in the RAM model), and a point can be inserted or deleted in $O(\log n)$ time [65]. Slightly weaker bounds are known in the pointer-machine model.

COLORED INTERSECTION SEARCHING

Preprocess a given set S of colored objects in $\mathbb{R}^d$ (i.e., each object in S is assigned a color) so that we can report (or count) the colors of the objects that intersect the query range. This problem arises in many contexts in which one wants to answer intersection-searching queries for nonconstant-size input objects. For example, given a set $P = \{P_1, \ldots, P_m\}$ of m simple polygons, one may wish to report all polygons of P that intersect a query segment; the goal is to return the indices, and not the description, of these polygons. If we color the edges of P_i with color i, the problem reduces to colored segment intersection searching in a set of segments.

A colored orthogonal range searching query for points on a two-dimensional grid $[0, U]^2$ can be answered in $O(\log\log U + k)$ time using $O(n \log^2 U)$ storage and $O(n \log n \log^2 U)$ preprocessing [8]. On the other hand, a set S of n colored rectangles in the plane can be stored into a data structure of size $O(n \log n)$ so that the colors of al rectangles in S that contain a query point can be reported in time $O(\log n + k)$ [18]. If the vertices of the rectangles in S and all the query points lie on the grid $[0, U]^2$, the query time can be improved to $O(\log\log U + k)$ by increasing the storage to $O(n^{1+\epsilon})$.

Gupta et al. [51] have shown that the colored halfplane-reporting queries in the plane can be answered in $O(\log^2 n + k)$ using $O(n \log n)$ space. Agarwal and van Kreveld [12] presented a linear-size data structure with $O(n^{1/2+\epsilon} + k)$ query time for colored segment intersection-reporting queries amid a set of segments in the plane, assuming that the segments of the same color form a connected planar graph, or if they form the boundary of a simple polygon; these data structures can also handle insertions of new segments.

31.6 OPTIMIZATION QUERIES

In optimization queries, we want to return an object that satisfies certain conditions with respect to a query range. The most common example of optimization queries is, perhaps, ray-shooting queries. Other examples include segment-dragging and linear-programming queries.

RAY-SHOOTING QUERIES

Preprocess a set S of objects in $\mathbb{R}^d$ into a data structure so that the first object (if one exists) intersected by a query ray can be reported efficiently. This problem arises in ray tracing, hidden-surface removal, radiosity, and other graphics problems. Efficient solutions to many geometric problems have also been developed using ray-shooting data structures.

A general approach to the ray-shooting problem, using segment intersection-detection structures and Megiddo's parametric-searching technique (Chapter 37), was proposed by Agarwal and Matoušek [9]. The basic idea of their approach is as follows. Suppose we have a segment intersection-detection data structure for S, based on partition trees. Let ρ be a query ray. Their algorithm maintains a segment $\vec{ab} \subseteq \rho$ so that the first intersection point of $\vec{ab}$ with S is the same as that of ρ. If a lies on an object of S, it returns a. Otherwise, it picks a point $c \in ab$ and determines, using the segment intersection-detection data structure, whether the interior of the segment ac intersects any object of S. If the answer is YES, it recursively finds the first intersection point of $\vec{ac}$ with S; otherwise, it recursively finds the first intersection point of $\vec{cb}$ with S. Using parametric searching, the point c at each stage can be chosen so that the algorithm terminates after $O(\log n)$ steps.

In some cases the query time can be improved by a polylogarithmic factor using a more direct approach.

TABLE 31.6.1 Ray shooting.

d	OBJECTS	$S(n)$	$Q(n)$
$d = 2$	simple polygon	n	$\log n$
	s disjoint polygons	n	$\sqrt{s} \log n$
	s disjoint polygons	$(s^2 + n) \log s$	$\log s \log n$
	s convex polygons	$sn \log s$	$\log s \log n$
	segments	m	$n/\sqrt{m}$
	circlular arcs	m	$n/m^{1/3}$
	disjoint arcs	n	$\sqrt{n}$
$d = 3$	convex polytope	n	$\log n$
	c-oriented polytopes	n	$\log n$
	s convex polytopes	$s^2 n^{2+\epsilon}$	$\log^2 n$
	halfplanes	m	$n/\sqrt{m}$
	terrain	m	$n/\sqrt{m}$
	triangles	m	$n/m^{1/4}$
	spheres	m	$n/m^{1/3}$
$d > 3$	hyperplanes	m	$n/m^{1/d}$
	hyperplanes	$\frac{n^d}{\log^{d-\epsilon} n}$	$\log n$
	convex polytope	m	$n/m^{1/\lfloor d/2 \rfloor}$

Table 31.6.1 gives a summary of known ray-shooting results. For the sake of clarity, we have ignored the polylogarithmic factors in the query time whenever it is of the form n/m^{α}.

Like simplex range searching, many practical data structures have been proposed that, despite having poor worst-case performance, work well in practice. One common approach is to construct a subdivision of $\mathbb{R}^d$ into constant-size cells so that the interior of each cell does not intersect any object of S. A ray-shooting query can be answered by traversing the query ray through the subdivision until we find an object that intersects the ray. The worst-case query time is proportional to the maximum number of cells intersected by a segment that does not intersect any object in S. Hershberger and Suri [54] showed that a triangulation with $O(\log n)$ query time can be constructed when S is the boundary of a simple polygon in the plane. Agarwal et al. [4] proved worst-case bounds for many cases on the number of cells in the subdivision that a line can intersect. Aronov and Fortune [15] have obtained a bound on the expected number of cells in the subdivision that a line can intersect.

LINEAR-PROGRAMMING QUERIES

Let S be a set of n halfspaces in $\mathbb{R}^d$. *We wish to preprocess S into a data structure so that for a direction vector $\vec{v}$, we can determine the first point of $\bigcap_{h \in S} h$ in the direction $\vec{v}$.* For $d \leq 3$, such a query can be answered in $O(\log n)$ time using $O(n)$ storage, by constructing the normal diagram of the convex polytope $\bigcap_{h \in S} h$ and preprocessing it for point-location queries. For higher dimensions, Ramos [71] has proposed two data structures. His first structure can answer a query in time $(\log n)^{O(\log d)}$ using $n^{\lfloor d/2 \rfloor} \log^{O(1)} n$ space and preprocessing, and his second structure can answer a query in time $n^{1-1/\lfloor d/2 \rfloor} 2^{O(\log^* n)}$ using $O(n)$ space and $O(n^{1+\epsilon})$ preprocessing.

SEGMENT-DRAGGING QUERIES

Preprocess a set S of objects in the plane so that for a query segment e and a ray ρ, the first position at which e intersects any object of S as it is translated (dragged) along ρ can be determined quickly. This query can be answered in $O((n/\sqrt{m}) \log^c n)$ time, with $O(m)$ storage, using segment-intersection searching structures and the parametric-search technique. Chazelle [22] gave a linear-size, $O(\log n)$ query-time data structure for the special case in which S is a set of points, e is a horizontal segment, and ρ is the vertical direction. Instead of dragging a segment along a ray, one can ask the same question for dragging along a more complex trajectory (along a curve, and allowing both translation and rotation). These problems arise naturally in motion planning and manufacturing.

31.7 SOURCES AND RELATED MATERIAL

RELATED READING

Books and Monographs

[64]: Multidimensional searching and computational geometry.

[35]: Basic topics in computational geometry.

[66]: Randomized techniques in computational geometry. Chapters 6 and 8 cover range-searching, intersection-searching, and ray-shooting data structures.

[30]: Covers lower bound techniques, ϵ-nets, cuttings, and simplex range searching.

[59, 74]: Range-searching data structures in spatial database systems.

Survey Papers

[6, 62]: Range-searching data structures.

[47, 68, 73] Indexing techniques used in databases.

[11]: Range-searching data structures for moving points.

[13]: Secondary-memory data structures.

[20]: Ray-shooting data structures.

RELATED CHAPTERS

Chapter 10: Geometric discrepancy theory and uniform distribution
Chapter 21: Arrangements
Chapter 30: Point location
Chapter 32: Ray shooting and lines in space

REFERENCES

[1] P. K. Agarwal, L. Arge, and J. Erickson. Indexing moving points. In *Proc. Annu. ACM Sympos. Principles Database Syst.*, 2000. 175–186.

[2] P. K. Agarwal, L. Arge, J. Erickson, P. G. Franciosa, and J. S. Vitter. Efficient searching with linear constraints. *Journal of Computer and System Sciences*, 61:194–216, 2000.

[3] P. K. Agarwal, L. Arge, O. Procopiuc, and J. S. Vitter. A framework for index bulk loading and dynamization. In *Proc. 28th Intl. Conference on Automata and Programming Langs.*, pages 115–127, 2001.

[4] P. K. Agarwal, B. Aronov, and S. Suri. Stabbing triangulations by lines in 3d. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 267–276, New York, NY, USA, June 1995. ACM Press.

[5] P. K. Agarwal, M. de Berg, J. Gudmundsson, M. Hammar, and H. J. Haverkort. Box-trees and r-trees with near-optimal query time. *Discrete Comput. Geom.*, 26:291–312, 2002.

[6] P. K. Agarwal and J. Erickson. Geometric range searching and its relatives. In B. Chazelle, J. E. Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry*, volume 223 of *Contemporary Mathematics*, pages 1–56. American Mathematical Society, Providence, RI, 1999.

[7] P. K. Agarwal, J. Gao, and L. Guibas. Kinetic medians and kd-trees. In *Proc. 10th European Sympos. Algorithms*, pages 15–26, 2002.

[8] P. K. Agarwal, S. Govindrajan, and S. Muthukrishnan. Range searching in categorical data: Colored range searching on grid. In *Proc. 10th European Sympos. Algorithms*, pages 17–28, 2002.

[9] P. K. Agarwal and J. Matoušek. Ray shooting and parametric search. *SIAM J. Comput.*, 22(4):794–806, 1993.

[10] P. K. Agarwal and J. Matoušek. On range searching with semialgebraic sets. *Discrete Comput. Geom.*, 11:393–418, 1994.

[11] P. K. Agarwal and C. M. Procopiuc. Advances in indexing mobile objects. *IEEE Bulletin of Data Engineering*, 25(2):25–34, 2002.

[12] P. K. Agarwal and M. van Kreveld. Polygon and connected component intersection searching. *Algorithmica*, 15:626–660, 1996.

[13] L. Arge. External memory data structures. In J. Abello, P. M. Pardalos, and M. G. C. Resende, editors, *Handbook of Massive Data Sets*, pages 313–358. Kluwer Academic Publishers, Boston, 2002.

[14] L. Arge, V. Samoladas, and J. S. Vitter. On two-dimensional indexability and optimal range search indexing. In *Proc. Annu. ACM Sympos. Principles Database Syst.*, pages 346–357, 1999.

[15] B. Aronov and S. Fortune. Approximating minimum-weight triangulations in three dimensions. *Discrete and Comput. Geom.*, 21:527–549, 1999.

[16] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, Sept. 1975.

[17] J. L. Bentley. Multidimensional divide-and-conquer. *Commun. ACM*, 23(4):214–229, 1980.

[18] P. Bozanis, N. Ktsios, C. Makris, and A. Tsakalidis. New results on intersection query problems. *The Computer Journal*, 40:22–29, 1997.

[19] H. Brönnimann, B. Chazelle, and J. Pach. How hard is halfspace range searching. *Discrete Comput. Geom.*, 10:143–155, 1993.

[20] A. Y. Chang. A survey of geometric data structures for ray tracing. Tech. Report TR-CIS-2001-06, Polytechnic University, 2001.

[21] B. Chazelle. Filtering search: a new approach to query-answering. *SIAM J. Comput.*, 15(3):703–724, 1986.

[22] B. Chazelle. An algorithm for segment-dragging and its implementation. *Algorithmica*, 3:205–221, 1988.

[23] B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM J. Comput.*, 17(3):427–462, June 1988.

[24] B. Chazelle. Lower bounds on the complexity of polytope range searching. *J. Amer. Math. Soc.*, 2:637–666, 1989.

[25] B. Chazelle. Lower bounds for orthogonal range searching, I: The reporting case. *J. ACM*, 37:200–212, 1990.

[26] B. Chazelle. Lower bounds for orthogonal range searching, II: The arithmetic model. *J. ACM*, 37:439–463, 1990.

[27] B. Chazelle. Cutting hyperplanes for divide-and-conquer. *Discrete Comput. Geom.*, 9(2):145–158, 1993.

[28] B. Chazelle. Lower bounds for off-line range searching. *Discrete Comput. Geom.*, 17(1):53–66, 1997.

[29] B. Chazelle. A spectral approach to lower bounds with applications to geometric searching. *SIAM J. Comput.*, 27(2):545–556, 1998.

[30] B. Chazelle. *The Discrepancy Method: Randomness and Complexity.* Cambridge University Press, New York, 2001.

[31] B. Chazelle and A. Lvov. A trace bound for hereditary discrepancy. *Discrete Comput. Geom.*, 26:221–232, 2001.

[32] B. Chazelle and B. Rosenberg. Computing partial sums in multidimensional arrays. In *Proc. 5th Annu. ACM Sympos. Comput. Geom.*, pages 131–139, 1989.

[33] B. Chazelle and B. Rosenberg. Simplex range reporting on a pointer machine. *Comput. Geom. Theory Appl.*, 5:237–247, 1996.

[34] B. Chazelle, M. Sharir, and E. Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. *Algorithmica*, 8:407–429, 1992.

[35] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry: Algorithms and Applications.* Springer-Verlag, Berlin, 1997.

[36] P. F. Dietz and R. Raman. Persistence, amortization and randomization. In *Proc. ACM-SIAM Sympos. Discrete Algorithms*, pages 78–88. 1991.

[37] D. P. Dobkin and H. Edelsbrunner. Space searching for intersecting objects. *J. Algorithms*, 8:348–361, 1987.

[38] J. Erickson. New lower bounds for halfspace emptiness. In *Proc. 37th Annu. IEEE Sympos. Found. Comput. Sci.*, pages 472–481, 1996.

[39] J. Erickson. New lower bounds for Hopcroft's problem. *Discrete Comput. Geom.*, 16:389–418, 1996.

[40] J. Erickson. Space-time tradeoffs for emptiness queries. *SIAM J. Computing*, 19:1968–1996, 2000.

[41] G. Evangelidis, D. B. Lomet, and B. Salzberg. The hB$^{\Pi}$-tree: A multi-attribute index supporting concurrency, recovery and node consolidation. *VLDB Journal*, 6:1–25, 1997.

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

[43] M. L. Fredman. The complexity of maintaining an array and computing its partial sums. *J. ACM*, 29:250–260, 1979.

[44] M. L. Fredman. The inherent complexity of dynamic data structures which accommodate range queries. In *Proc. 21st Annu. IEEE Sympos. Found. Comput. Sci.*, pages 191–199, 1980.

[45] M. L. Fredman. A lower bound on the complexity of orthogonal range queries. *J. ACM*, 28:696–705, 1981.

[46] M. L. Fredman. Lower bounds on the complexity of some optimal data structures. *SIAM J. Comput.*, 10:1–10, 1981.

[47] V. Gaede and O. Günther. Multidimensional access methods. *ACM Comput. Surv.*, 30:170–231, 1998.

[48] S. Govindarajan, P. K. Agarwal, and L. Arge. CRB-tree: An efficient indexing scheme for range aggregate queries. In *Proc. 9th Intl. Conf. Database Theory*, 2003.

[49] J. Gray, A. Bosworth, A. Layman, and H. Patel. Data cube: A relational aggregation operator generalizing group-by, cross-tab, and sub-totals. In *Proc. 12th IEEE Internat. Conf. Data Eng.*, pages 152–159, 1996.

[50] L. J. Guibas. Kinetic data structures — a state of the art report. In P. K. Agarwal, L. E. Kavraki, and M. Mason, editors, *Proc. Workshop Algorithmic Found. Robot.*, pages 191–209. A. K. Peters, Wellesley, MA, 1998.

[51] P. Gupta, R. Janardan, and M. Smid. Efficient algorithms for generalized intersection searching on non-iso-oriented objects. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 369–378, 1994.

[52] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. ACM SIGMOD Conf. Principles Database Systems*, pages 47–57, 1984.

[53] D. Haussler and E. Welzl. Epsilon-nets and simplex range queries. *Discrete Comput. Geom.*, 2:127–151, 1987.

[54] J. Hershberger and S. Suri. A pedestrian approach to ray shooting: Shoot a ray, take a walk. *J. Algorithms*, 18:403–431, 1995.

[55] C.-T. Ho, R. Agrawal, N. Megiddo, and R. Srikant. Range queries in OLAP data cubes. In *Proc. ACM SIGMOD Conf. on Management of Data*, pages 73–88, 1997.

[56] C.-T. Ho, J. Bruck, and R. Agrawal. Partial-sum queries in OLAP data cubes using covering codes. In *Proc. Annu. ACM Sympos. Principles Database Syst.*, pages 228–237, 1997.

[57] V. Koltun. Almost tight upper bounds for vertical decompositions in four dimensions. In *Proc. 42nd Sympos. on Foundations of Computer Science*, pages 56–65, 2001.

[58] D. B. Lomet and B. Salzberg. The hB-tree: A multiattribute indexing method with good guaranteed performance. *ACM Trans. Database Syst.*, 15:625–658, 1990.

[59] Y. Manolopoulos, Y. Theodoridis, and V. Tsotras. *Advanced Database Indexing.* Kluwer Academic Publishers, Boston, 1999.

[60] J. Matoušek. Efficient partition trees. *Discrete Comput. Geom.*, 8:315–334, 1992.

[61] J. Matoušek. Range searching with efficient hierarchical cuttings. *Discrete Comput. Geom.*, 10(2):157–182, 1993.

[62] J. Matoušek. Geometric range searching. *ACM Comput. Surv.*, 26:421–461, 1994.

[63] E. M. McCreight. Priority search trees. *SIAM J. Comput.*, 14(2):257–276, 1985.

[64] K. Mehlhorn. *Data Structures and Algorithms 3: Multi-dimensional Searching and Computational Geometry*, volume 3 of *EATCS Monographs on Theoretical Computer Science.* Springer-Verlag, Heidelberg, Germany, 1984.

[65] C. W. Mortensen. Fully-dynamic two dimensional orthogonal range and line segment intersection reporting in logarithmic time. In *Proc. 14th ACM-SIAM Sympos. Discrete Algorithms*, 2003.

[66] K. Mulmuley. *Computational Geometry: An Introduction Through Randomized Algorithms.* Prentice Hall, Englewood Cliffs, NJ, 1993.

[67] J. Nievergelt, H. Hinterberger, and K. C. Sevcik. The grid file: An adaptable, symmetric multi-key file structure. *ACM Trans. Database Syst.*, 9(1):38–71, 1984.

[68] J. Nievergelt and P. Widmayer. Spatial data structures: Concepts and design choices. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, pages 725–764. Elsevier Science Publishers B.V. North-Holland, Amsterdam, 2000.

[69] M. H. Overmars. *The Design of Dynamic Data Structures*, volume 156 of *Lecture Notes Comput. Sci.* Springer-Verlag, Heidelberg, West Germany, 1983.

[70] C. M. Procopiuc, P. K. Agarwal, and S. Har-Peled. Star-tree: An efficent self-adjusting index for moving points. In *Proc. 4th Workshop on Algorithm Engineering and Experiments*, pages 178–193, 2002.

[71] E. A. Ramos. Linear programming queries revisited. In *Proc. 16th Annu. ACM Sympos. Comput. Geom.*, pages 176–181, 2000.

[72] J. T. Robinson. The k-d-b-tree: a search structure for large multidimensional dynamic indexes. In *Proc. ACM SIGACT-SIGMOD Conf. Principles Database Systems*, pages 10–18, 1981.

[73] B. Salzberg and V. J. Tsotras. A comparison of access methods for time evolving data. *ACM Comput. Surv.*, 31(2):158–221, 1999.

[74] H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison-Wesley, Reading, MA, 1990.

[75] T. Sellis, N. Roussopoulos, and C. Faloutsos. The R$^+$-tree: A dynamic index for multi-dimensional objects. In *Proc. 13th VLDB Conference*, pages 507–517, 1987.

[76] S. Šaltenis, C. S. Jensen, S. T. Leutenegger, and M. A. Lopez. Indexing the positions of continuously moving objects. In *Proc. ACM SIGMOD International Conference on Management of Data*, pages 331–342, 2000.

[77] J. Vuillemin. A unifying look at data structures. *Commun. ACM*, 23:229–239, 1980.

[78] D. Willard. Lower bounds for the addition-subtraction operations in orthogonal range queries and related problems. *Inform. Comput.*, 82:45–64, 1989.

[79] D. E. Willard. Polygon retrieval. *SIAM J. Comput.*, 11:149–165, 1982.

[80] A. C. Yao. Space-time trade-off for answering range queries. In *Proc. 14th Annu. ACM Sympos. Theory Comput.*, pages 128–136, 1982.

[81] A. C. Yao. On the complexity of maintaining partial sums. *SIAM J. Comput.*, 14:277–288, 1985.