41 RANGE SEARCHING

Pankaj K. Agarwal

INTRODUCTION

A central problem in computational geometry, range searching 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 R be a family of subsets of $\mathbb{R}^d$; elements of R are called ***ranges***. Typical examples of ranges include rectangles, halfspaces, simplices, and balls. *Preprocess S into a data structure so that for a query range $\gamma \in \mathsf{R}$, the points in $S \cap \gamma$ can be reported or counted efficiently.* A single query can be answered in linear time using linear space, by simply checking for each point of S whether it lies in the query range. Most of the applications, however, call for querying the same set S numerous times, in which case it is desirable 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 *range-emptiness queries*: determine whether $S \cap \gamma = \emptyset$; and *range-min/max queries*: each point has a weight and one must return the point in the query range with the minimum/maximum weight. Many different types of range-searching queries can be encompassed in the following general formulation of the range-searching problem:

Let $(\mathbf{S}, +)$ be a commutative semigroup. Each point $p \in S$ is assigned a weight $w(p) \in \mathbf{S}$. For any subset $S' \subseteq S$, let $w(S') = \sum_{p \in S'} w(p)$, where addition is taken over the semigroup. For a query range $\gamma \in \mathsf{R}$, 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 range-max queries by choosing the semigroup to be $(\mathbb{R}, \max)$ and choosing $w(p)$ to be the weight of p.

A more general (decomposable) *geometric searching* problem can be defined as follows: 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, R a set of ranges, and $\Diamond \subseteq S \times \mathsf{R}$ a "spatial" relation between objects and ranges. For a query range $\gamma \in \mathsf{R}$, the goal is to compute $\sum_{p \Diamond \gamma} w(p)$. Range searching is a special case of this geometric 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* and denoted by $Q(n)$; by the *size* of the data structure, denoted by $S(n)$; and by the time spent in constructing in the data structure, called the *preprocessing time* and denoted by $P(n)$. Since the data structure is constructed only once, its query time and size are generally more important than its preprocessing time. If a data structure supports insertion and deletion operations, its *update time* is also relevant. 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 chapter, k will be used to denote the output size.

We assume d to be a small constant, and big-O and big-Ω notation hide constants depending on d. The dependence on d of the performance of almost all the data structures mentioned in this chapter 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. Assuming the coordinates of input points to be real numbers, the query time in any reasonable model of computation such as pointer machines, RAM, or algebraic decision trees is $\Omega(\log n)$ even when $d = 1$ (faster query time is possible if the coordinates are integers, say, bounded by $n^{O(1)}$). Therefore, a natural question is whether a linear-size data structure with logarithmic query time exists for range searching. 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, one seeks a trade-off 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 space/query-time trade-off can be achieved?

41.1 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, if the coordinates are integers in the range $[0{:}U]$,[1] for some $U \geq n$, then memory cells can contain arbitrary $\omega := O(\log U)$ bit long integers, called *words*, which can be added, multiplied, subtracted, divided (computing $\lfloor x/y \rfloor$), compared, and used as pointers to other memory cells in constant time. The real RAM model allows each memory cell to store arbitrary real numbers, and it supports constant-time arithmetic and relational operations between two real numbers, though conversions between integers and reals are not allowed. In the case of range searching over a semigroup other than integers, memory cells are allowed 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 the RAM and the 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 out-degree 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

[1]For $b \geq a$, we use $[a{:}b]$ to denote the set of integers between a and b.

a portion of the graph and visits at least one node with label i for each point p_i in the query range. Chazelle [47] defines several generalizations of the pointer-machine model that are more appropriate for answering counting and semigroup queries. In these models, nodes are labeled with arbitrary $O(\log n)$-bit integers, and the query algorithm is allowed to perform arithmetic operations on these integers.

The ***cell probe model*** is the most basic model for proving lower bounds on data structures [128]. In this model, a data structure consists of a set of memory cells, each storing ω bits. Each cell is identified by an integer address, which fits in ω bits. The data structure answers a query by probing a number of cells from the data structure and returns the correct answer based on the contents of the probed cells. It handles an update operation by reading and updating (probing) a number of cells to reflect the changes. The cost of an operation is the number of cells probed by the data structure to perform that operation.

The best lower bound on the query time one can hope to prove in the cell probe model is $\Omega(\text{polylog}(n))$, which is far from the best-known upper bounds. Extensive work has been done on proving lower bounds in the ***semigroup arithmetic model***, originally introduced by Fredman [72] and refined by Yao [130]. In this model, a data structure can be regarded informally 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.

The informal model we have just described is much too powerful. For example, the optimal data structure for range-counting queries in this semigroup model consists of the $n+1$ integers $0, 1, \ldots, n$. A counting query can be answered by simply returning the correct answer. Since no additions are required, a query can be answered in zero "time," using a linear-size data structure. The notion of ***faithful*** semigroup circumvents this problem: A commutative semigroup $(\mathbf{S}, +)$ is faithful if for each $n > 0$, for any sets of indices $I, J \subseteq [1{:}n]$ 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, 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 an object 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 $(S, \mathbf{S}, \mathsf{R}, \Diamond)$ is a collection of generators $\{g_1, g_2, \ldots, g_s\}$ with the following property: For any query range $\gamma \in \mathsf{R}$, there is a set of indices $I_\gamma \subseteq [1:s]$ and a set of labeled nonnegative integers $\{\beta_i \mid i \in I_\gamma\}$ such that the linear forms $\sum_{p_i \Diamond \gamma} x_i$ and $\sum_{i\in I_\gamma} \beta_i g_i$ 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.

A serious weakness of the semigroup model is that it does not allow subtractions even if the weights of points belong to a group (e.g. range counting). Therefore the ***group model*** has been proposed in which each point is assigned a weight from a commutative group and the goal is to compute the group sum of the weights of points lying in a query range. The data structure consists of a collection of group elements and auxiliary data, and it answers a query by adding and subtracting a subset of the precomputed group elements to yield the answer to the query. The query time is the number of group operations performed [74, 53].

The lower-bound proofs in the semigroup model have a strong geometric flavor because subtractions are not allowed: the query algorithm can use a precomputed sum that involves the weight of a point p only if p lies in the query range. A typical proof basically reduces to arguing that not all query ranges can be "covered" with a small number of subsets of input objects [54]. Unfortunately, no such property holds for the group model, which makes proving lower bounds in the group model much harder. Notwithstanding recent progress, the known lower bounds in the group model are much weaker than those under the semigroup model.

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. Queries are answered with such a data structure by performing a depth-first search through the resulting recursive space partition. The ***partition-graph*** model, introduced by Erickson [65, 66], formalizes this divide-and-conquer approach. This model can be used to study the complexity of emptiness queries, which are trivial in semigroup and pointer-machine models.

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 $\mathsf{F} = \mathsf{F}(S)$ of *canonical subsets* of S and stores the weight $w(C) = \sum_{p \in C} w(p)$ of each canonical subset $C \in \mathsf{F}$. For a query range γ, the query procedure determines a partition $\mathsf{C}_\gamma = \mathsf{C}(S, \gamma) \subseteq \mathsf{F}$ of $S \cap \gamma$ and adds the weights of the subsets in C_γ to compute $w(S \cap \gamma)$. We 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 model described earlier. Each canonical subset $C = \{p_i \mid i \in I\} \in \mathsf{F}$, where $I \subseteq [1 : n]$, corresponds to the generator $\sum_{i \in I} x_i$. How exactly the weights of canonical subsets are stored and how C_γ 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 C_γ, 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 C_γ. In a *hierarchical* decomposition scheme, the canonical subsets and their weights are organized in a tree T. Each node v of T is associated with a canonical subset $C_v \in \mathsf{F}$, and if w is a child of v in T then $C_w \subseteq C_v$. Besides the weight of C_v, some auxiliary information is also stored at v, which is used to determine whether $C_v \in \mathsf{C}_\gamma$ for a query range γ. If the weight of each canonical subset can be stored in $O(1)$ memory cells and if one can determine in $O(1)$ time whether $C_w \in \mathsf{C}_\gamma$ where w is a descendent of a given node v, the hierarchical decomposition scheme is called *efficient*. The total size of an efficient decomposition scheme is simply $O(|\mathsf{F}|)$. For range-reporting queries, in which the "weight" of a canonical subset is the set itself, the size of the data structure is $O(\sum_{C\in\mathsf{F}}|C|)$, but it can be reduced to $O(|\mathsf{F}|)$ by storing the canonical subsets implicitly. Finally, let $r > 1$ be a parameter, and set $\mathsf{F}_i = \{C \in \mathsf{F} \mid r^{i-1} \le |C| \le r^i\}$. A hierarchical decomposition scheme is called *r-convergent* if there exist constants $\alpha \ge 1$ and $\beta > 0$ so that the degree of every node in T is $O(r^\alpha)$ and for all $i \ge 1$, $|\mathsf{F}_i| = O((n/r^i)^\alpha)$ and, for all query ranges γ, $|\mathsf{C}_\gamma \cap \mathsf{F}_i| = O((n/r^i)^\beta)$, 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 41.5 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 C_γ (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 $1 \le i \le a$, form a partition of $C_v \cap \gamma$. Then the procedure searches each w_i recursively. The total query time is $O(\log n + |\mathsf{C}_\gamma|)$ if the decomposition scheme is r-convergent for some constant $r > 1$ and constant time is spent at each node visited.

41.2 ORTHOGONAL RANGE SEARCHING

An abstraction of multi-key searching in database systems, ranges are axis-aligned rectangles in d-dimensional orthogonal range 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. A query of the form—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 an orthogonal range-reporting query.

UPPER BOUNDS

Most orthogonal range-searching data structures with polylog(n) query time under the pointer-machine model are based on ***range trees***, introduced by Bentley [35]. 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 $R = [a_1, b_1] \times [a_2, b_2]$ can be answered by traversing T in a top-down manner as follows. Suppose a node v is being visited by the query procedure. If v is a leaf and R contains the point of C_v, then report the point. 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 $[a_v, b_v] \subseteq [a_1, b_1]$, report all the points of

C_v whose y-coordinates lie in the interval $[a_2, b_2]$, by performing a binary search. Otherwise, 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. A d-dimensional range tree can be extended to $\mathbb{R}^{d+1}$ by using a multi-level structure (see Section 41.5) and by paying a $\log n$ factor in storage and query time.

Since the original range-tree paper by Bentley, several data structures with improved bounds have been proposed; see Table 41.2.1 for a summary of known results. If queries are "3-sided rectangles" in $\mathbb{R}^2$, say, of the form $[a_1, b_1] \times [a_2, \infty)$, then a ***priority search tree*** of size $O(n)$ can answer a range-reporting query in time $O(\log n + k)$ [100]. Chazelle [46] showed that an orthogonal range reporting query in $\mathbb{R}^2$ can be answered in $O(\log n + k)$ time using $O(n\mathrm{Log}n)$ space, where $\mathrm{Log}n = \log_{\log n} n = \log n / \log\log n$, by constructing a range tree of $O(\log n)$ fanout and storing additional auxiliary structures at each node. For $d = 3$, Afshani *et al.* [4] proposed a data structure of $O(n\mathrm{Log}^2 n)$ size with $O(\log n + k)$ query time. Afshani *et al.* [3, 4] showed that a range-tree based d-dimensional range searching data structure can be extended to $\mathbb{R}^{d+1}$ by paying a cost of $\mathrm{Log}n$ in both space and query time. For $d = 4$, a query thus can be answered in $O(\log n \mathrm{Log}n + k)$ time using $O(n\mathrm{Log}^3 n)$ space. Afshani *et al.* [5] presented a slightly different data structure of size $O(n\mathrm{Log}^4 n)$ that answers a 4D query in $O(\log n \sqrt{\mathrm{Log}n} + k)$ time. For $d \geq 4$, a query can be answered in $O(\log(n)\mathrm{Log}^{d-3} n + k)$ time using $O(n\mathrm{Log}^{d-1} n)$ space, or in $O(\log(n)\mathrm{Log}^{d-4+1/(d-2)} n + k)$ time using $O(n\mathrm{Log}^d n)$ space. The preprocessing time of these data structures is $O(n \log^2 n\mathrm{Log}^{d-3} n)$ and $O(n \log^3 n\mathrm{Log}^{d-3} n)$, respectively [8]. More refined bounds on the query time can be found in [4].

There is extensive work on range-reporting data structures in the RAM model when $S \subset [0{:}U]^d$ for some integer $U \geq n$, assuming that each memory cell can store a word of length $\omega = O(\log U)$. For $d = 1$, a range-reporting query can be answered in $O(\log \omega + k)$ time using the van Emde Boas tree of linear size [121]. Alstrup *et al.* [24] proposed a linear-size data structure that can answer a range reporting query in $O(k + 1)$ time. In contrast, any data structure of size $n^{O(1)}$ for finding the predecessor in S of a query point has $\Omega(\log \omega)$ query time [23].

Using standard techniques, the universe can be reduced to $[0{:}n]^d$ by paying $O(\log\log U)$ additional time in answering a query, so for $d \geq 2$, we assume that $U = n$ and that the coordinates of the query rectangle are also integers in the range $[0{:}n]$; we refer to this setting as the *rank space*.

For $d = 2$, the best known approach can answer a range-reporting query in $O((1 + k)\log\log n)$ time using $O(n \log\log n)$ space, in $O((1 + k)\log^\epsilon n)$ time using $O(n)$ space, or in $O(\log\log n + k)$ time using $O(n \log^\epsilon n)$ space [41], where $\epsilon > 0$ is an arbitrarily small constant. For $d = 3$, a query can be answered in $O(\log\log n + k)$ time using $O(n \log^{1+\epsilon} n)$ space; if the query range is an octant, then the size of the data structure is only linear. These data structures can be extended to higher dimensions by paying a factor of $\log^{1+\epsilon} n$ in space and $\mathrm{Log}n$ in query time per dimension. Therefore a d-dimensional query can be answered in $O(\mathrm{Log}^{d-3} n \log\log n + k)$ using $O(n \log^{d-2+\epsilon} n)$ space.

The classical range-tree data structure can answer a 2D range-counting query in $O(\log n)$ time using $O(n \log n)$ space. Chazelle [47] showed that the space can be improved to linear in the generalized pointer-machine or the RAM model by using a compressed range tree. For $d \geq 2$, by extending Chazelle's technique, a range-counting query in the rank space can be answered in $O(\mathrm{Log}^{d-1} n)$ time using $O(n\mathrm{Log}^{d-2} n)$ space [84]. Such a data structure can be constructed in time $O(n \log^{d-2+1/d} n)$ [42]. Chan and Wilkinson [44] presented an adaptive data structure of $O(n \log\log n)$ size that can answer a 2D range-counting query in $O(\log\log n + \mathrm{Log} k)$, where k is the number of points in the query range. Their technique can also be used to answer range-counting queries approximately: it returns a number k' with $k \le k' \le (1+\delta)k$, for a given constant $\delta > 0$, in $O(\log\log n)$ time using $O(n \log\log n)$ space, or in $O(\log^{\epsilon} n)$ time using $O(n)$ space. If an additive error of ϵn is allowed, then an approximate range-counting query can be answered using $O(\frac{1}{\epsilon}\log(\frac{1}{\epsilon})\log\log(\frac{1}{\epsilon})\log n)$ space, and this bound is tight within $O(\log\log\frac{1}{\epsilon})$ factor [126].

In an off-line setting, where all queries are given in advance, n d-dimensional range-counting queries can be answered in $O(n \log^{d-2+1/d} n)$ time [42].

All the data structures discussed above can be dynamized using the standard partial-rebuilding technique [106]. 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 de-amortization techniques [63]. Faster dynamic data structures have been developed for a few cases, especially under the RAM model when $S \subseteq [0{:}U]^d$ and $d = 1, 2$. For example, Mortensen *et al.* [101] dynamized the 1D range-reporting data structure with $O(\log\omega)$ update time and $O(\log\log\omega + k)$ query time.

TABLE 41.2.1 Orthogonal range reporting upper bounds; $h \in [1, \log^{\epsilon} n]$.

d	Pointer Machine		RAM	
	$S(n)$	$Q(n)$	$S(n)$	$Q(n)$
$d = 1$	n	$\log n + k$	n	$1 + k$
$d = 2$	$n\frac{\log n}{\log\log n}$	$\log n + k$	$nh \log\log n$ $n \log_h \log n$	$\log\log n + k \log_h \log n$ $(1+k)h \log\log n$
$d = 3$	$n(\frac{\log n}{\log\log n})^2$	$\log n + k$	$n \log^{1+\epsilon} n$	$\log\log n + k$
$d \ge 4$	$n(\frac{\log n}{\log\log n})^{d-1}$ $n(\frac{\log n}{\log\log n})^{d}$	$\frac{\log^{d-2} n}{\log^{d-3}\log n} + k$ $\frac{\log^{\alpha+1} n}{\log^{\alpha}\log n} + k$ $\alpha = d - 4 + \frac{1}{d-2}$	$n \log^{d-2+\epsilon} n$	$\frac{\log^{d-3} n}{\log^{d-4}\log n} + k$

LOWER BOUNDS

Semigroup model. Fredman [71, 72] was the first to prove nontrivial lower bounds on orthogonal range searching, but in a dynamic setting in which points can be inserted and deleted. He showed that a mixed sequence of n insertions, deletions, and queries takes $\Omega(n \log^d n)$ time under the semigroup model. Chazelle [50] proved lower bounds for the static version of orthogonal range searching, which almost match the best upper bounds known. He showed that there exists a set S of n weighted points in $\mathbb{R}^d$, with weights from a faithful semigroup, such that the worst-case query time, under the semigroup model, for an orthogonal range-searching data structure of size nh is $\Omega((\log_h n)^{d-1})$. If the data structure also supports insertions of points, the query time is $\Omega((\log_h n)^d)$. These lower bounds hold even if the queries are orthants instead of rectangles, i.e., for the so-called ***dominance***

queries. In fact, they apply to answering the dominance query for a randomly chosen query point. It should be pointed out that the bounds in [50] assume the weights of points in S to be a part of the input, i.e., the data structure is not tailored to a special 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.

Group model. Pătraşcu [108] proved that a dynamic data structure in the group model for 2D dominance counting queries that supports updates in expected time t_u requires $\Omega((\frac{\log n}{\log(t_u \log n)})^2)$ expected time to answer a query. Larsen [89] proved lower bounds for dynamic range searching data structures in the group model in terms of combinatorial discrepancy of the underlying set system. For d-dimensional orthogonal range searching, any dynamic data structure with the worst-case t_u and t_q update and query time, respectively, requires $t_u t_q = \Omega(\log_\omega^{d-1} n)$ [90].

Cell-probe model. Pătraşcu [108, 109] and Larsen [88] also proved lower bounds on orthogonal range searching in the cell probe model. In particular, Pătraşcu proved that a data structure for 2D dominance counting queries that uses nh space requires $\Omega(\log_{h\omega} n)$ query time, where $\omega = \Omega(\log n)$ is the size of each memory cell. He also proved the same lower bound for 4D orthogonal range reporting. Note that for $h = n\text{polylog}n$ and $\omega = \log n$, the query time is $\Omega(\text{Log}n)$ using $O(n\text{polylog}n)$ space, which almost matches the best known upper bound of $O(\log n + k)$ using $O(n \log^{2+\epsilon} n)$ space. In contrast, a 3D orthogonal range query can be answered in $O(\log\log n + k)$ time using $O(n \log^{1+\epsilon} n)$ space.

Larsen [88] proved that a dynamic data structure for the 2D weighted range counting problem with t_u worst-case update time has $\Omega((\log_{\omega t_u} n)^2)$ expected query time; the weight of each point is an $O(\log n)$ bit integer. Note that for $t_u = \text{polylog}(n)$ and $\omega = \Theta(\log n)$, the expected query time is $\Omega(\text{Log}^2 n)$.

Pointer-machine model. For range reporting queries, Chazelle [49] proved that the size of any data structure on a pointer machine that answers a d-dimensional range-reporting query in $O(\text{polylog}(n) + k)$ time is $\Omega(n\text{Log}^{d-1}n)$. Notice that this lower bound is greater than the known upper bound for answering two-dimensional reporting queries on the RAM model. Afshani *et al.* [4, 5] adapted Chazelle's technique to show that a data structure for range-reporting queries that uses nh space requires $\Omega(\log(n) \log_h^{\lfloor d/2 \rfloor - 2} n + k)$ time to answer a query.

Off-line searching. 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 [52] proved that the off-line version takes $\Omega(n\text{Log}^{d-1}n)$ time in the semigroup model and $\Omega(n \log\log n)$ time in the group model. For $d = \Omega(\log n)$ (resp. $d = \Omega(\text{Log}n)$), the lower bound for the off-line range-searching problem in the group model can be improved to $\Omega(n \log n)$ (resp. $\Omega(n\text{Log}n)$) [56].

PRACTICAL 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.

Many practical data structures construct a recursive partition of space, typically into rectangles, and a tree induced by this partition. The simplest example of this type of data structure is the ***quad tree*** [116]. A quad tree in $\mathbb{R}^2$ is a 4-way tree, each of whose nodes v 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. The ***compressed quad tree*** guarantees its size to be linear, though the depth can be linear in the worst case [116].

Quad trees and their variants construct a grid on a square 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 of Bentley [34]. 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.

Inserting/deleting points dynamically into a kd-tree is expensive, so 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 alternatives of kd-trees have been proposed to optimize space and query time, most notably ***buddy tree*** [117] and ***hB-tree*** [92, 68]. A buddy tree is a combination of quad- and kd-trees in the sense that rectangles are split into sub-rectangles only at some specific locations, which simplifies the split procedure. In an ***hB-tree***, the region associated with a node is allowed to be $R_1 \setminus R_2$ where R_1 and R_2 are rectangles. The same idea has been used in the ***BBd-tree***, a data structure developed for answering approximate nearest-neighbor and range queries [30].

Several extensions of kd-trees, such as random projection trees [60], randomized partition trees [61], randomly oriented kd-trees [122], and cover trees [36], have been proposed to answer queries on a set S of points in $\mathbb{R}^d$ when s is very high but the intrinsic dimension of S is low. A popular notion of intrinsic dimension of S is its *doubling dimension*, i.e., the smallest integer b such that for every $x \in \mathbb{R}^d$ and for every $r > 0$, the points of S within distance r from x can be covered by 2^b balls of radius $r/2$. It has been shown that the perforamce of these data structures on certain queries depends expoentially only on the doubling dimension of S.

PARTIAL-SUM QUERIES

Partial-sum queries require preprocessing a d-dimensional array A with n entries, in an additive semigroup, into a data structure, so that for a d-dimensional rectangle

$\gamma = [a_1, b_1] \times \cdots \times [a_d, b_d]$, 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 off-line version, given A and m rectangles $\gamma_1, \gamma_2, \ldots, \gamma_m$, we wish to compute $\sigma(A, \gamma_i)$ for each i. 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 called ***data cube*** [77] that represents the data as a d-dimensional array. An aggregate query on a data cube 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; see [83, 123] and the references therein.

If the sum is a group operation, then the query can be answered in $O(1)$ time in any fixed dimension by maintaining the prefix sums and using the inclusion-exclusion principle. Yao [129] showed that, for $d = 1$, a partial-sum query where sum is a semigroup operation can be answered in $O(\alpha(n))$ time using $O(n)$ space; here $\alpha(n)$ is the inverse Ackermann function. If the sum operation is *max* or *min*, then a partial-sum query can be answered in $O(1)$ time under the RAM model [70, 125].

For $d > 1$, Chazelle and Rosenberg [57] 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 off-line 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 the values in the array can be updated, then Fredman [74] proved a lower bound of $\Omega(\log n / \log\log n)$ on the maximum of update and query time. The bound was later improved to $\Omega(\log n)$ by Pătraşcu and Demaine [110].

RANGE-STATISTICS QUERIES

The previous subsections focused on two versions of range searching—reporting and counting. In many applications, especially when the input data is large, neither of the two is quite satisfactory — there are too many points in the query range to report, and a simple count (or weighted sum) of the number of points gives too little information. There is recent work on computing some statistics on points lying in a query rectangle, mostly for $d = 1, 2$.

Let S be a set of weighted points. The simplest range statistics query is the *range min/max* query: return a point of the minimum/maximum weight in a query rectangle. If the coordinates of input points are real numbers, then the classical range tree can answer a range min/max query in time $O(\log^{d-1} n)$ using $O(n \log^{d-1} n)$ space. Faster data structures have been developed for answering range-min/max queries in the rank space. For $d = 1$, as mentioned above, a range min/max query can be answered in $O(1)$ time using $O(n)$ space in the RAM model. For $d = 2$, a range min/max query can be answered in $O(\log\log n)$ time using $O(n \log^{\epsilon} n)$ space [41].

Recently, there has been some work on the *range-selection* query for $d = 1$: let $S = [1{:}n]$ and w_i be the weight of point i. Given an interval $[i{:}j]$ and an integer $r \in [1{:}j - i + 1]$, return the r-th smallest value in $w_i, \ldots, w_j$. Chan and

Wilkinson [44] have described a data structure of linear size that can answer the query in time $O(1+\log_\omega r)$ in the ω-bit RAM model, and Jørgensen and Larsen [86] proved that this bound is tight in the worst case, i.e., any data structure that uses nh space, where $h \geq 1$ is a parameter, requires $\Omega(\log_{\omega h} n)$ time to answer a query. For $d \geq 2$, Chan and Zhou [45] showed that a range-selection query can be answered in $O((\frac{\log n}{\log\log n})^d)$ time using $O(n(\frac{\log n}{\log\log n})^{d-1})$ space.

A 1D *range-mode* query, i.e., given an interval $[i{:}j]$ return the mode of $w_i, \ldots, w_j$, can be computed in $O(\sqrt{n/\log n})$ time [40]. A close relationship between range-mode queries and Boolean matrix multiplication implies that any data structure for range-mode queries must have either $\Omega(n^{t/2})$ preprocessing time or $\Omega(n^{t/2-1})$ query time in the worst case, where t is the exponent in the running time of a matrix-multiplication algorithm; the best known matrix-multiplication algorithm has exponent 2.3727.

Rahul and Janardan [112] considered the problem of reporting the set of maximal points (i.e., the points that are not dominated by any other point) in a query range, the so-called *skyline query.* They presented a data structure of size $O(n\log^{d+1} n)$ that can report all maximal points in a query rectangle in time $O(k\log^{d+2} n)$. For $d = 2$, Brodal and Larsen [38] show that a skyline-reporting query in the rank space can be answered in $O(\frac{\log n}{\log\log n} + k)$ time using $O(n\log^\epsilon n)$ space or in $O(\frac{\log n}{\log\log n} + k\log\log n)$ time using $O(n\log\log n)$ space. They also propose a linear-size data structure that answers a skyline-counting query in $O(\frac{\log n}{\log\log n})$ time, and show that this bound is optimal in the worst case.

OPEN PROBLEMS

1. For $d > 2$, prove a lower bound on the query time for orthogonal range counting that depends on d.
2. Can a range-reporting query for $d = 4, 5$ under the pointer-machine model be answered in $O(\log n + k)$ time using $n\text{polylog}(n))$ space?

41.3 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) What is the size of a data structure that answers simplex range queries in logarithmic time? (ii) How fast can a simplex range query be answered using a linear-size data structure?

GLOSSARY

Arrangement: The arrangement of a set H of hyperplanes in $\mathbb{R}^d$, denoted by $\mathsf{A}(H)$, is the subdivision of $\mathbb{R}^d$ into cells, each being a maximal connected set contained in the intersection of a fixed subset of H and not intersecting any other hyperplane of H.

Level: The level of a point p with respect to a set H of hyperplanes is the number of hyperplanes of H that lie below p.

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 $(b_1, \ldots, b_{d-1}, b_d)$.

$1/r$-cutting: Let H be a set of n hyperplanes in $\mathbb{R}^d$, and let $r \in [1, n]$ be a parameter. A $(1/r)$-cutting of H is a set Ξ of (relatively open) disjoint simplices covering $\mathbb{R}^d$ so that at most n/r hyperplanes of H cross (i.e., intersect but do not contain) each simplex of Ξ.

Shallow cutting: Let H be a set of n hyperplanes in $\mathbb{R}^d$, and let $r \in [1, n], q \in [0{:}n-1]$ be two parameters. A shallow $(1/r)$-cutting of H is a set Ξ of (relatively open) disjoint simplices covering all points with level (with respect to H) at most q so that at most n/r hyperplanes of H cross each simplex of Ξ.

DATA STRUCTURES WITH LOGARITHMIC QUERY TIME

For simplicity, consider the following *halfspace range-counting* problem: Preprocess a set S of n points into a data structure so that the number of points of S that lie above a query hyeprplane can be counted quickly. Using the duality transform, the above halfspace range-counting problem can be reduced to the following point-location problem: *Given a set H of n hyperplanes, determine the number of hyperplanes of H lying above a query point q.* Since the same subset of hyperplanes lies above all points of a single cell of $\mathsf{A}(H)$, the number of hyperplanes of H lying above q can be reported by locating the cell of $\mathsf{A}(H)$ that contains q. The following theorem of Chazelle [51] leads to an $O(\log n)$ query-time data structure for halfspace range counting.

THEOREM 41.3.1 *Chazelle* [51]

Let H be a set of n hyperplanes and $r \le n$ a parameter. Set $s = \lceil \log_2 r \rceil$. There exist s 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 $O(nr^{d-1})$.

Choose $r = \lceil \frac{n}{\log_2 n} \rceil$. Construct the cuttings $\Xi_1, \ldots, \Xi_s$, for $s = \lceil \log_2 r \rceil$; for each simplex $\triangle \in \Xi_i$, for $i < s$, store pointers to the simplices of Ξ_{i+1} that are contained in $\triangle$; and for each simplex $\triangle \in \Xi_s$, store $H_\triangle \subseteq H$, the set of hyperplanes that intersect $\triangle$, and $k_\triangle$, the number of hyperplanes of H that lie above $\triangle$. Since $|\Xi_s| = O(r^d)$, the total size of the data structure is $O(nr^{d-1}) = O(n^d/\log^{d-1} n)$. For a query point $q \in \mathbb{R}^d$, by traversing the pointers, find the simplex $\triangle \in \Xi_s$ that contains q, count the number of hyperplanes of $H_\triangle$ that lie above q, and return $k_\triangle$ plus this quantity. The total query time is $O(\log n)$.

The above approach can be extended to the *simplex range-counting* problem: 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. [58] proposed a data structure of size $O(n^{d+\epsilon})$, for any $\epsilon > 0$, using a multi-level data structure (see Section 41.5), that can answer a simplex range-counting query in $O(\log n)$ time. The space bound can be reduced to $O(n^d)$ by increasing the query time to $O(\log^{d+1} n)$ [96].

LINEAR-SIZE DATA STRUCTURES

Most of the linear-size data structures for simplex range searching are based on ***partition trees***, originally introduced by Willard [127] for a set of points in the plane. Roughly speaking, a partition tree is a hierarchical decomposition scheme (in the sense described in Section 41.1) 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 41.1. The query time depends on the maximum number of regions associated the with the children 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 $O(n^\alpha + k)$, where $\alpha = \log_4 3 \approx 0.793$. A major breakthrough in simplex range searching was made by Haussler and Welzl [81]. 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, was first given by Matoušek [96] and subsequently simplified by Chan [39]. These data structures answer a simplex range-counting (resp. range-reporting) query in $\mathbb{R}^d$ in time $O(n^{1-1/d})$ (resp. $O(n^{1-1/d} + k)$), and are based on the following theorem.

THEOREM 41.3.2 *Matoušek* [94]

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 $i \neq j$, 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 41.3.2. 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 $O(n^{1-1/d+\log_r c} + k)$; the $\log_r c$ term in the exponent 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 41.3.2 that builds a simplicial partition hierarchically analogous to Theorem 41.3.1, instead of building it at each level independently, leads to a linear-size data structure with $O(n^{1-1/d} + k)$ query time. Chan's (randomized) algorithm [39] constructs a hierarchical simplicial partition in which the (relative) interiors of simplices at every level are pairwise-disjoint and they together induce a hierarchical partition of $\mathbb{R}^d$. A space/query-time trade-off for simplex range searching can be attained by combining the linear-size and logarithmic query-time data structures [18].

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 $O(n^{1-\gamma+\epsilon})$ using linear space, where $\gamma = 1/\lfloor (d+b)/2 \rfloor$.

HALFSPACE RANGE REPORTING

A halfspace range-reporting query can be answered more quickly than a simplex range-reporting query using shallow cutting. For simplicity, throughout this subsection we assume the query halfspace to lie below its bounding hyperplane. We begin by discussing a simpler problem: the *halfspace-emptiness* query, which asks whether a query halfspace contains any input point. By the duality transform, the halfspace-emptiness query in $\mathbb{R}^d$ can be formulated as asking whether a query point $q \in \mathbb{R}^d$ lies below all hyperplanes in a given set H of hyperplanes in $\mathbb{R}^d$. This query is equivalent to asking whether q lies inside a convex polyhedron $\mathsf{P}(H)$, defined by the intersection of halfspaces lying below the hyperplanes of H. For $d \leq 3$, a point-location query in $\mathsf{P}(H)$ can be answered optimally in $O(\log n)$ time using $O(n)$ space and $O(n \log n)$ preprocessing since $\mathsf{P}(H)$ has linear size [62]. For $d \geq 4$, point-location query in $\mathsf{P}(H)$ becomes more challenging, and the query is answered using a shallow-cutting based data structure. The following theorem by Matoušek can be used to construct a point-location data structure for $\mathsf{P}(H)$:

THEOREM 41.3.3 *Matoušek* [95]

Let H be a set of n hyperplanes and $r \leq n$ a parameter. A shallow $(1/r)$-cutting of level 0 *with respect to H of size $O(r^{\lfloor d/2 \rfloor})$ can be computed in time $O(nr^c)$, where c is a constant depending on d.*

Choose r to be a sufficiently large constant, and compute a shallow $(1/r)$-cutting Ξ of level 0 of H using Theorem 41.3.3. For each simplex $\triangle \in \Xi$, let $H_\triangle \subseteq H$ be the set of hyperplanes that intersect $\triangle$. Recursively, construct the data structure for $H_\triangle$; the recursion stops when $|H_\triangle| \leq r$. The size of the data structure is $O(n^{\lfloor d/2 \rfloor + \epsilon})$, where $\epsilon > 0$ is an arbitrarily small constant, and it can be constructed in $O(n^{\lfloor d/2 \rfloor + \epsilon})$ time. If a query point q does not lie in a simplex of Ξ, then one can conclude that $q \notin \mathsf{P}(H)$ and thus stop. Otherwise, if q lies in a simplex $\triangle \in \Xi$, recursively determine whether q lies below all the hyperplanes of $H_\triangle$. The query time is $O(\log n)$. Matoušek and Schwarzkopf [99] showed that the space can be reduced to $O(\frac{n^{\lfloor d/2 \rfloor}}{\log^{\lfloor d/2 \rfloor - \epsilon} n})$.

A linear-size data structure can be constructed for answering halfspace-emptiness queries by constructing a simplicial partition analogous to Theorem 41.3.2 but with the property that a hyperplane that has at most n/r points above it crosses $O(r^{1-1/\lfloor d/2 \rfloor})$ simplices. By choosing r appropriately, a linear-size data structure can be constructed in $O(n^{1+\epsilon})$ time that answers a query in $n^{1-1/\lfloor d/2\rfloor}2^{O(\log^* n)}$ time; the construction time can be reduced to $O(n\log n)$ at the cost of increasing the query time to $O(n^{1-1/\lfloor d/2\rfloor}\text{polylog}(n))$. For even dimensions, a linear-size data structure with query time $O(n^{1-1/\lfloor d/2\rfloor})$ can be constructed in $O(n\log n)$ randomized-expected time [39].

The halfspace-emptiness data structure can be adapted to answer halfspace range-reporting queries. For $d = 2$, Chazelle *et al.* [55] presented an optimal data structure with $O(\log n + k)$ query time, $O(n)$ space, and $O(n\log n)$ preprocessing time. For $d = 3$, after a series of papers with successive improvements, a linear-size data structure with $O(\log n + k)$ query times was proposed by Afshani and Chan [6]; this structure can be constructed in $O(n\log n)$ time [43]. Table 41.3.1 summarizes the best known bounds for halfspace range reporting in higher dimensions; halfspace-reporting data structures can also answer halfspace-emptiness queries without the output size term in their query time.

TABLE 41.3.1 Near-linear-size data structures for halfspace range reporting/emptiness.

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

Finally, we comment that halfspace-emptiness data structures have been adapted to answer halfspace range-counting queries approximately. For example, a set S of n points in $\mathbb{R}^3$ can be preprocessed, in $O(n\log n)$ time, into a linear-size data structure that for a query halfspace γ in $\mathbb{R}^3$, can report in $O(\log n)$ time a number t such that $|\gamma\cap S| \le t \le (1+\delta)|\gamma\cap S|$, where $\delta > 0$ is a constant [6, 7]. For $d > 3$, such a query can be answered in $O((\frac{n}{t})^{1-1/\lfloor d/2\rfloor}\text{polylog}(n))$ time using linear space [111].

LOWER BOUNDS

Fredman [73] 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, under the semigroup model. His technique, however, does not extend to static data structures. In a seminal paper, Chazelle [48] proved an almost matching lower bound on simplex range searching under the semigroup model. He showed that any data structure of size m, for $n \le m \le n^d$, for simplex range searching in the semigroup model requires a query time of $\Omega(n/\sqrt{m})$ for $d = 2$ and $\Omega(n/(m^{1/d}\log n))$ for $d \ge 3$ in the worst case. His lower bound holds even if the query ranges are wedges or strips. For halfspaces, Arya *et al.* [31] proved a lower bound of $\Omega((\frac{n}{\log n}^{1-\frac{1}{d+1}} m^{-\frac{1}{d+1}}))$ on the query time under the semigroup model. They also showed that if the semigroup is integral (i.e., for all non-zero elements of the semigroup and for all $k \geq 2$, the k-fold sum $x + \cdots + x \neq x$), then the lower bound can be improved to $\Omega(\frac{n}{m^{1/d}} \log^{-1-\frac{2}{d}} n)$.

A few lower bounds on simplex range searching have been proved under the group model. Chazelle [53] proved an $\Omega(n \log n)$ lower bound for off-line halfspace range searching under the group model. Exploiting a close-connection between range searching and discrepancy theory, Larsen [89] showed that for any dynamic data structure with t_u and t_q update and query time, respectively, $t_u \cdot t_q = \Omega(n^{1-1/d})$.

The best-known lower bound for simplex range reporting in the pointer-machine model is by Afshani [2] who proved that the size of any data structure that answers a simplex range reporting query in time $O(t_q + k)$ is $\Omega((\frac{n}{t_q})^d / 2^{O(\sqrt{\log t_q})})$. His technique also shows that the size of any halfspace range-reporting data structure in dimension $d(d+3)/2$ has size $\Omega((\frac{n}{t_q})^d / 2^{O(\sqrt{\log t_q})})$.

A series of papers by Erickson established the first nontrivial lower bounds for on-line and off-line 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 $\Omega(n \log m + n^{2/3} m^{2/3} + m \log n)$ [66], 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 [96]. He later established lower bounds on a trade-off between space and query time, or preprocessing and query time, for on-line hyperplane emptiness queries [67]. 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 $O(n\text{polylog}n)$ preprocessing time is allowed. For $d = 2$, if the preprocessing time is t_p, the query time is $\Omega(n/\sqrt{t_p})$.

OPEN PROBLEMS

1. Prove a near-optimal lower bound on static simplex range searching in the group model.
2. Prove an optimal lower bound on halfspace range reporting in the pointer-machine model.
3. Can a halfspace range counting query be answered more efficiently if query hyerplanes satisfy certain properties, e.g., they are tangent to $\mathbb{S}^{d-1}$?

41.4 SEMIALGEBRAIC RANGE SEARCHING

GLOSSARY

Semialgebraic set: A semialgebraic set is a subset of $\mathbb{R}^d$ obtained from a finite number of sets of the form $\{x \in \mathbb{R}^d \mid g(x) \geq 0\}$, where g is a d-variate polynomial with real coefficients, by Boolean operations (union, intersection, and complement). A semialgebraic set has *constant description complexity* if the dimension, the number of polynomials defining the set, as well as the maximum degree of these polynomials are all constants.

Tarski cell: A semialgebraic cell of constant description complexity

Partiotning polynomial: For a set $S \subset \mathbb{R}^d$ of n points and a real parameter r, $1 < r \leq n$, an r-partitioning polynomial for S is a nonzero d-variate polynomial f such that each connected component of $\mathbb{R}^d \setminus Z(f)$ contains at most n/r points of S, where $Z(f) := \{x \in \mathbb{R}^d \mid f(x) = 0\}$ denotes the zero set of f. The decomposition of $\mathbb{R}^d$ into $Z(f)$ and the connected components of $\mathbb{R}^d \setminus Z(f)$ is called a *polynomial partition* (induced by f).

So far we have assumed the ranges to be bounded by hyperplanes, but many applications require ranges to be defined by non-linear 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 with balls as ranges. A more general class of ranges can be defined as follows.

Let $\Gamma_{d,\Delta,s}$ denote the family of all semialgebraic sets in $\mathbb{R}^d$ defined by at most s polynomial inequalities of degree at most Δ each. The range-searching problem in which query ranges belong to $\Gamma_{d,\Delta,s}$ for constants d, Δ, s, is referred to as *semialgebraic range searching.*

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 therefore assume the ranges to be 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 describe two approaches for answering such queries.

TABLE 41.4.1 Semialgebraic range searching; λ is the dimension of linearization.

d	RANGE	$S(n)$	$Q(n)$	NOTES
$d = 2$	disk	n	$\sqrt{n}\log n$	Counting
			$\log n + k$	Reporting
	Tarski cell	n	$\sqrt{n}\log^c n$	Counting
$d \geq 3$	ball	n	$n^{1-\frac{1}{d}+\epsilon}$	Counting
			$n^{1-\frac{1}{\lceil d/2 \rceil}}\log^c n + k$	Reporting
$d = 2t - 1$	ball	n	$n^{1-\frac{1}{t}} + k$	Reporting
$d \geq 3$	Tarski cell	n	$n^{1-1/d}\log^c n$	Counting
			$n^{1-\frac{1}{\lambda}+\epsilon}$	

LINEARIZATION

One approach to answering Γ_f-range queries is the ***linearization*** method. The polynomial $f(x, a)$ is represented 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 $\Gamma_f(a) = \{x \in \mathbb{R}^d \mid f(x, a) \geq 0\}$ maps to a halfspace

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

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 [18] 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.

ALGEBRAIC METHOD

Agarwal and Matoušek [18] had also proposed an approach for answering Γ_f-range queries, by extending Theorem 41.3.2 to Tarski cells and by constructing partition trees using this extension. The query time of this approach depends on the complexity of the so-called *vertical decomposition* of arrangements of surfaces, and it leads to suboptimal performance for $d > 4$. A better data structure has been proposed [19, 98] based on the *polynomial partitioning scheme* introduced by Guth and Katz [79]; see Chapter 7.

Let $S \subset \mathbb{R}^d$ be a set of n points, and let r, $1 < r \leq n$, be a real parameter. Guth and Katz show that an r-partitioning polynomial of degree $O(r^{1/d})$ for S always exists. Agarwal *et al.* [19] described a randomized algorithm to compute such a polynomial in expected time $O(nr + r^3)$. A result by Barone and Basu [33] implies that an algebraic variety of dimension k defined by polynomials of constant-bounded degree crosses $O(r^{k/d})$ components of $\mathbb{R}^d \setminus Z(f)$, and that these components can be computed in time $r^{O(1)}$. Therefore, one can recursively construct the data structure for points lying in each component of $\mathbb{R}^d \setminus Z(f)$. The total time spent in recursively searching in the components crossed by a query range will be $n^{1-1/d}\text{polylog}(n)$. However, this ignores the points in $S^* = S \cap Z(f)$. Agarwal *et al.* [19] use a scheme based on the so-called cylindrical algebraic decomposition to handle S^*. A more elegant and simpler method was subsequently proposed by Matoušek and Patáková [98], which basically applies a generalized polynomial-partitioning scheme on S^* and $Z(f)$. Putting everything together, a semilagebraic range-counting query can be answered in $O(n^{1-1/d}\text{polylog}(n))$ time using a linear-size data structure; all k points lying inside the query range can be reported by spending an additional $O(k)$ time.

Arya and Mount [29] have presented a linear-size data structure for approximate range-searching queries. Let γ be a constant-complexity semialgebraic set and $\epsilon > 0$

a parameter. Their data structure returns in $O(\frac{1}{\epsilon^d}\log n + k_\epsilon)$ time a subset S_ϵ of k_ϵ points such that $\gamma \cap S \subseteq S_\epsilon \subseteq \gamma_\epsilon \cap S$ where γ_ϵ is the set of points within distance $\epsilon \cdot \mathrm{diam}(\gamma)$ of γ. If γ is convex, the query time improves to $O(\log n + \frac{1}{\epsilon^{d-1}} + k_\epsilon)$. A result by Larsen and Nguyen [90] implies that query time of a linear-size data structure is $\Omega(\log n + \epsilon^{-\frac{d}{1+\delta}-1})$ for any arbitrarily small constant $\delta > 0$. The data structure in [29] can also return a value k_ϵ, with $|S \cap \gamma| \le k_\epsilon \le |S \cap \gamma_\epsilon|$ in time $O(\frac{1}{\epsilon^d}\log n)$, or in $O(\log n + \frac{1}{\epsilon^{d-1}})$ time if γ is convex.

41.5 VARIANTS AND EXTENSIONS

In this section we review a few extensions of range-searching data structures: multi-level structures, secondary-memory structures, range searching in a streaming model, range searching on moving points, and coping with data uncertainty.

MULTI-LEVEL STRUCTURES

A powerful property of data structures based on decomposition schemes (described in Section 41.1) 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 [64], 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 P, with underlying relation $\Diamond$, requires computing $\sum_{p \Diamond \gamma} w(p)$ for a query range γ. Let P^1 and P^2 be two geometric-searching problems, and let $\Diamond^1$ and $\Diamond^2$ be the corresponding relations. Define $\mathsf{P}^1 \circ \mathsf{P}^2$ to be the conjunction of P^1 and P^2, whose relation is $\Diamond^1 \cap \Diamond^2$. For a query range γ, the goal is to compute $\sum_{p \Diamond^1 \gamma, p \Diamond^2 \gamma} w(p)$. Suppose there are hierarchical decomposition schemes D^1 and D^2 for problems P^1 and P^2. Let $\mathsf{F}^1 = \mathsf{F}^1(S)$ be the set of canonical subsets constructed by D^1, and for a range γ, let $\mathsf{C}^1_\gamma = \mathsf{C}^1(S,\gamma)$ be the corresponding partition of $\{p \in S \mid p \Diamond^1 \gamma\}$ into canonical subsets. For each canonical subset $C \in \mathsf{F}^1$, let $\mathsf{F}^2(C)$ be the collection of canonical subsets of C constructed by D^2, and let $\mathsf{C}^2(C,\gamma)$ be the corresponding partition of $\{p \in C \mid p \Diamond^2 \gamma\}$ into level-two canonical subsets. The decomposition scheme $\mathsf{D}^1 \circ \mathsf{D}^2$ for the problem $\mathsf{P}^1 \circ \mathsf{P}^2$ consists of the canonical subsets $\mathsf{F} = \bigcup_{C \in \mathsf{F}^1} \mathsf{F}^2(C)$. For a query range γ, the query output is $\mathsf{C}_\gamma = \bigcup_{C \in \mathsf{C}^1_\gamma} \mathsf{C}^2(C,\gamma)$. Any number of decomposition schemes can be cascaded in this manner.

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

Suppose the size and query time of each decomposition scheme are at most $S(n)$

and $Q(n)$, respectively, and D^1 is efficient and r-convergent (cf. Section 41.1), for some constant $r > 1$. Then the size and query time of the decomposition scheme D are $O(S(n)\log_r n)$ and $O(Q(n)\log_r n)$, respectively. If D^2 is also efficient and r-convergent, then D is 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. Range trees for orthogonal range searching, logarithmic query-time data structures for simplex range searching, and data structures for semialgebraic range searching discussed above are a few examples of multi-level structures. More examples will be mentioned in the following sections.

COLORED RANGE SEARCHING

In *colored range searching* (or *categorical* range searching), each input point is associated with a color and the goal is to report or count the number of colors of points lying inside a query range. For $d = 1$, a colored (orthogonal) range-reporting query can be answered in $O(\log n + k)$ time using linear space, where k is now the number of colors reported [85]. For $d = 2$, a 3-sided-rectangle reporting query can be answered using $O(n)$ space in $O(\log n + k)$ time under the pointer-machine model, or in $O(\log\log n + k)$ time in the rank space under the RAM model [91]. These data structures extend to reporting the colors of points inside a (4-sided) rectangle in the same time but using $O(n \log n)$ space. Using the techniques in [12], colored halfplane-reporting queries in the plane can be answered in $O(\log n + k)$ time using $O(n)$ space. In general, a range-emptiness data structure with $S(n)$ space and $Q(n)$ query time can be extended to answer colored version of the range-reporting query for the same ranges in $O((1+k)Q(n))$ time and $O(S(n)\log n)$ space; if $S(n) = \Omega(n^{1+\epsilon})$, the size remains $O(S(n))$.

It has been shown that a colored range counting query in $\mathbb{R}^1$ is equivalent to (uncolored) range counting in $\mathbb{R}^2$ [85, 91], so a 1D colored range-counting query can be answered in $O(\log n)$ time under the pointer-machine model or in $O(\frac{\log n}{\log\log n})$ time under the RAM model and rank space, using linear space. For $d = 2$, by establishing a connection between colored orthogonal range counting and Boolean matrix multiplication, Kaplan *et al.* [87] showed that the query time of a 2D colored range counting is $\Omega(n^{t/2-1})$ where n^t is the time taken by a Boolean matrix multiplication algorithm. They also showed that a d-dimensional colored orthogonal range counting query can be answered in $O(\log^{2d-1} n)$ time using $O(n^d \log^{2d-1} n)$ space; if the query ranges are d-dimensional orthants, then the query time and space can be improved to $O(n^{d-1}\log n)$ and $O(n^{\lfloor d/2\rfloor}\log^{d-1} n)$, respectively. Finally, we note that an approximate counting query can be answered in $O(\log^{d+1} n)$ time using $O(n \log^{d+1} n)$ space [111].

SECONDARY MEMORY STRUCTURES

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* ***I/O model***, in which main memory has finite size, secondary memory is unlimited, and data is stored in secondary memory in blocks of size B, where B is a parameter [22]. Each access to secondary memory transfers one block (i.e., B words), and we count this as one *input/output (I/O) operation.* 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 range-reporting query time is $\Omega(\log_B n + k/B)$. There have been various extensions of this model, including the so-called ***cache-oblivious model*** in which the value of B is not known and the goal is to minimize I/O as well as the total work performed.

I/O-efficient range-searching structures have received much attention because of large data sets in spatial databases. The main idea underlying these structures is to construct high-degree trees instead of binary trees. For example, B-trees and their variants are used to answer 1-dimensional range-reporting queries in $O(\log_B n + \kappa)$ I/Os [116], where $\kappa = k/B$. Similarly, the *kdB*-tree, an I/O-efficient version of kd-tree, was proposed to answer high-dimensional orthogonal range queries [114]. While storing each point only once, it can answer a d-dimensional range-reporting query using $O((n/B)^{1-1/d} + \kappa)$ I/Os.

Arge et al. [27] developed an external priority search tree so that a 2D 3-sided rectangle-reporting query can be answered in $O(\log_B n + \kappa)$ I/Os using $O(n)$ space. 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. In contrast, a data structure in the cache-oblivious model that answers a 3-sided query in $O(\mathrm{polylog}(n) + \kappa)$ time needs $\Omega(n(\log n)^{\epsilon})$ space, and the same holds for 3D halfspace range reporting queries [9]. By extending the ideas proposed in [49], it can be shown that any I/O-efficient data structure that answers a range-reporting query using $O(\log_B^c n + \kappa)$ I/Os requires $\Omega(n \log_B n / \log \log_B n)$ storage. Table 41.5.1 summarizes the best known bounds on range reporting queries in the I/O and cache-oblivious models.

By extending the data structure in [29], Streppel and Yi [119] have presented a linear-size I/O-efficient data structure for approximate range reporting. For a constant complexity range γ, it returns using $O(\frac{1}{\epsilon^d} \log_B n + k_\epsilon/B)$ I/Os, a subset S_ϵ of k_ϵ points such that $S \cap \gamma \subseteq S_\epsilon \subseteq S \cap \gamma_\epsilon$, where γ_ϵ is the set of points in $\mathbb{R}^d$ within distance $\epsilon \cdot \mathrm{diam}(\gamma)$ from γ.

Govindrajan et al. [76] have shown that a two-dimensional orthogonal range counting query can be answered in $O(\log_B n)$ I/Os using linear space, assuming that each word can store $\log n$ bits. As for internal memory data structures, I/O-range-emptiness data structures can be adapted to answer range-counting queries approximately. For example, a 3D halfspace or dominance range-counting query can be answered approximately in $O(\log_B n)$ I/Os using linear space in the cache-oblivious model [7]. Colored range searching also has been studied in the I/O model, and efficient data structures are known for $d \leq 2$ [91, 104]: a colored orthogonal range-reporting query in $\mathbb{R}^2$ can be answered in $O(\log_B n + k/B)$ I/Os using $O(n \log(n) \log^* n)$ space.

Perhaps the most widely used I/O-efficient data structure for range searhing in higher dimensions is the ***R-tree***, originally introduced by Guttman [80]. An R-tree

TABLE 41.5.1 Secondary-memory structures for orthogonal range searching.

d	RANGE	MODEL	$Q(n)$	$S(n)$
$d=1$	interval	C.O.	$\log_B n + \kappa$	n
$d=2$	3-sided	I/O	$\log_B n + \kappa$	n
		C.O.	$\log_B n + \kappa$	$n\sqrt{\log n}$
	rectangle	I/O	$\log_B n + \kappa$	$\frac{n \log n}{\log \log_B n}$
		C.O.	$\log_B n + \kappa$	$n \log^{3/2} n$
	halfplane	I/O	$\log_B n + \kappa$	n
	triangle	I/O	$\sqrt{n/B} + \kappa$	n
$d=3$	octant	I/O	$\log_B n + \kappa$	n
		C.O.	$\log_B n + \kappa$	$n \log n$
	box	I/O	$\log_B n + \kappa$	$n(\frac{\log n}{\log \log_B n})^3$
		C.O	$\log_B n + \kappa$	$n \log^{7/2} n$
	halfspace	I/O	$\log_B n + \kappa$	$n \log^* n$
		C.O.	$\log_B n + \kappa$	$n \log n$
$d \geq 3$	box	I/O	$\log_B n(\frac{\log n}{\log \log_B n})^{d-2} + \kappa$	$n(\frac{\log n}{\log \log_B n})^{d-1}$
	simplex	I/O	$(n/B)^{1-1/d} + \kappa$	n

is a B-tree, each of whose nodes stores a set of rectangles. Each leaf stores a subset of input points, 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 an R-tree so that the overlap is minimized. Several heuristics for improving the performance, including R*- and Hilbert-R-trees, have been proposed though the query time is linear in the worst case. Agarwal et al. [13] showed how to construct a variant of the R-tree, called the *box 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. Arge *et al.* [26] adapted their method to define a version of R-tree with the same query time.

STREAMING MODEL

Motivated by a broad spectrum of applications, data streams have emerged as an important paradigm for processing data that arrives on-line. In many such applications, data is too large to be stored in its entirety or to even scan for real-time processing. The goal is therefore to construct a small-size summary of the data stream (arrived so far) that can be used to analyze or query the data. The monograph by Muthukrishnan [103] gives a summary of algorithms developed in the streaming model.

In the context of range searching, a few algorithms have been proposed for constructing a "succinct" data structure so that a range-counting query can be answered approximately, roughly with additive error ϵn. All the proposed data structures are based on the notion of ϵ-approximation: For a parameter $\epsilon > 0$, a subset $A \subseteq S$ is an *ϵ-approximation* for a set R of ranges if for every $\gamma \in \mathsf{R}$,

$$\left| \frac{|S \cap \gamma|}{|S|} - \frac{|A \cap \gamma|}{|A|} \right| \leq \epsilon.$$

A more general result by Li *et al.* [**?**] implies that a random subset of size $O(\frac{1}{\epsilon^2} \log \frac{1}{\delta})$ is an ϵ-approximation with probability at least $1-\delta$ for geometric ranges of constant complexity. Better bounds are known for many cases. For example, an ϵ-approximation of size $O(\frac{1}{\epsilon}\text{polylog}(\frac{1}{\epsilon}))$ exists for rectangles in $\mathbb{R}^d$, and of size $O(\epsilon^{-\frac{2d}{d+1}})$ for halfspaces in $\mathbb{R}^d$.

Bagchi *et al.* [32] described a deterministic algorithm for maintaining an ϵ-approximation of size $O(\frac{1}{\epsilon^2} \log \frac{1}{\epsilon})$ for a large class of ranges. It uses $O(\frac{1}{\epsilon^{2(d-1)}}\text{polylog}(n/\epsilon))$ space, and uses the same amount of time to update the ϵ-approximation when a new point arrives. Faster algorithms are known for special cases. For $d = 1$, an ϵ-approximation of size $O(\frac{1}{\epsilon} \log n)$ with respect to a set of intervals can be maintained efficiently by a deterministic algorithm in the streaming model [78]. The space can be improved to $O(\frac{1}{\epsilon} \log \frac{1}{\epsilon})$ if one allows a randomized algorithm [69]. For $d \geq 2$, an ϵ-approximation of size $O(\frac{1}{\epsilon} \log^{2d+1} \frac{1}{\epsilon})$ for rectangles and of size $O(\epsilon^{-\frac{2d}{d+1}} \log^{d+1} \frac{1}{\epsilon})$ for halfspaces can be maintained efficiently [120].

KINETIC RANGE SEARCHING

Let $S = \{p_1, \ldots, p_n\}$ be a set of n points in $\mathbb{R}^2$, each moving continuously with fixed velocity. Let $p_i(t) = a_i + b_i t$, for $a_i, b_i \in \mathbb{R}^2$, denote the position of p_i at time t, and let $S(t) = \{p_1(t), \ldots, p_n(t)\}$. 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 focus on 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, known as the *time-oblivious* approach, regards time as a new dimension and stores the trajectories $\bar{p}_i$ of input points p_i. An advantage of this 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 $\mathbb{R}^3$ 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. Using a 2D simplex range-reporting data structure, L can be preprocessed into a linear-size data structure so that all lines intersecting σ can be reported in $O(\sqrt{n} + k)$ time. A similar structure can answer Q2 queries within the same asymptotic time bound. The lower bounds on simplex range searching suggest that this approach will not lead to a near-linear-size data structure with $O(\log n + k)$ query time.

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). 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. A two-level partition tree of size $O(n)$ can report, in $O(n^{1/2+\epsilon} + k)$ time, all k lines of L intersecting R [10]. Again a Q2 query can be answered within the same time bound.

The second approach, based on the ***kinetic-data-structure*** framework, builds a dynamic data structure on the moving points (see Chapter 54). 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 and is updated 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 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 balanced binary search 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 [10].

Since range trees are complicated, a more practical approach is to use the kinetic-data-structure framework on *kd*-trees, as proposed by Agarwal et al. [15]. They propose two variants 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. A variant of *kd*-tree with a slightly better performance was proposed in [1]. Kinetic *kd*-trees process too many events because of their strong invariants, kinetic R-trees have also been proposed [124, 107], which require weaker invariants and thus process fewer events.

RANGE SEARCHING UNDER UNCERTAINTY

Many applications call for answering range queries in the presence of uncertainty in data – the location of each point may be represented as a probability density function (pdf) or a discrete mass function, called *location uncertainty*, or each point may exist with certain probability, called *existential uncertainty.* In the presence of location uncertainty, the goal is to report the points that lie inside a query range with probability at least τ, for some parameter $\tau \in [0, 1]$, or count the number of such points. In the case of existential uncertainty, the goal is to return some statistics on the distribution of the points inside a query range, e.g., what is the probability distribution of the number of points inside a query range, or what is the expected/most-likely value of the maximum weight of a point inside a query range.

If the location of each point is given as a piecewise-constant pdf in $\mathbb{R}^1$, a data structure of Agarwal et. al. [12] reports all k points that lie inside a query interval with probability at least τ. For fixed τ, the query time is $O(\log n + k)$ and the size of the data structure is $O(n)$. If τ is part of the query, then the query time and size are $O(\log^3 n + k)$ and $O(n \log^2 n)$, respectively. They also describe another data stucture of near-linear size that can handle more general pdfs and answer fixed-τ queries in $O(\log n + k)$ time.

Agarwal *et al.* [16] have described a data structure for range-max/min queries under both existential and location uncertainty in $\mathbb{R}^d$. For $d = 1$, using $O(n^{3/2})$ space, their data structure can compute the most-likely or the expected value of the maximum weight of input points in a query interval in time $O(n^{1/2})$; their data structure extends to higher dimensions. They also present another data structure that can estimate the expected value of the maximum inside a query rectangle within factor 2 in $O(\text{polylog}(n))$ time using $O(n\text{polylog}(n))$ space.

OPEN PROBLEMS

1. Prove tight bounds on orthogonal range searching in higher dimensions in the I/O model.
2. 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?
3. How quickly can 2D range queries be answered under location uncertainty?
4. How quickly can 1D range-max query be answered under existential/location uncertainty in data using linear space?

41.6 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 $w : S \to \mathbf{S}$; preprocess S into a data structure so that for a query object γ, the weighted sum $\sum w(p)$, taken over all objects of S that intersect γ, can be computed quickly.* 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 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 set $\psi(\gamma)$ so that p intersects γ if and only if $\varphi(p) \in \psi(\gamma)$. For example, suppose both S and the query ranges are sets of segments in $\mathbb{R}^2$. 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 set $\psi(\gamma)$ 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, 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 (described in Section 41.5) 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. A two-level data structure can be constructed to answer such a query—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 structure, and hence a 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.

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 41.6.1 gives some of the known results. Counting data structures can also report the objects containing a query point, in an additional time proportional to the output size.

TABLE 41.6.1 Point intersection searching.

d	OBJECTS	$S(n)$	$Q(n)$	NOTES
$d = 2$	rectangles	n	$\log n + k$	reporting
	disks	m	$(n/\sqrt{m})^{4/3}$	counting
		n	$\log n + k$	reporting
	triangles	m	$\frac{n}{\sqrt{m}}\log^3 n$	counting
	fat triangles	$n\log^* n$	$\log n + k$	reporting
	Tarski cells	$n^{2+\epsilon}$	$\log n$	counting
$d = 3$	halfspaces	n	$\log n + k$	reporting
	Tarski cells	$n^{3+\epsilon}$	$\log n$	counting
$d \geq 3$	rectangles	$n\log^{d-2} n$	$\log^{d-1} n + k$	reporting
		$n\log^{d-2+\epsilon} n$	$\log n(\frac{\log n}{\log\log n})^{d-2} + k$	
	simplices	m	$\frac{n}{m^{1/d}}\log^{d+1} n$	counting
	balls	$n^{d+\epsilon}$	$\log n$	counting
		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 S 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 41.6.2 for some of the known results on segment intersection searching; polylogarithmic factors are omitted from the query-search time whenever it is of the form n/m^α.

TABLE 41.6.2 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

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 the edges of P_i are colored with color i, the problem reduces to colored segment intersection searching in a set of segments.

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 all rectangles in S that contain a query point can be reported in time $O(\log n + k)$ [37]. 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 $O(n^{1+\epsilon})$.

Agarwal and van Kreveld [21] presented a linear-size data structure with $O(n^{1/2+\epsilon} + k)$ query time for colored segment intersection-reporting queries amidst a set of segments in the plane, assuming that the segments of the same color form a connected planar graph or the boundary of a simple polygon.

41.7 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. Originally motivated by the ray-tracing problem in computer graphics, this problem has found many applications and has been studied extensively in computational geometry.

A general approach to the ray-shooting problem, using segment intersection-detection structures and Megiddo's parametric-searching technique, was proposed by Agarwal and Matoušek [17]. The basic idea of their approach is as follows: suppose there is a segment intersection-detection data structure for S, based on partition trees. Let ρ be a query ray. The query procedure maintains a segment $\vec{ab} \subseteq \rho$ that contains the first intersection point of ρ with S. 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 polylog(n) factor using a more direct approach. Table 41.7.1 gives a summary of known ray-shooting results; polylogarithmic factors are ignored in the query time whenever it is of the form n/m^{α}.

TABLE 41.7.1 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	$n/\sqrt{m}$
	circular 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	$s^2n^{2+\epsilon}$	$\log^2 n$
	fat convex polytopes	m	$n/\sqrt{m}$
	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	$n/m^{1/d}$
		$\frac{n^d}{\log^{d-\epsilon} n}$	$\log n$
	convex polytope	m	$n/m^{1/\lfloor d/2\rfloor}$
		$\frac{n^{\lfloor d/2\rfloor}}{\log^{\lfloor d/2\rfloor-\epsilon} n}$	$\log n$

Practical data structures have been proposed that, notwithstanding 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 [82] 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. [11] proved worst-case bounds for many cases on the number of cells in a subdivision in $\mathbb{R}^3$ that a line can intersect. Aronov and Fortune [28] have obtained a bound on the expected number of cells in the subdivision in $\mathbb{R}^3$ that a line can intersect.

ARC-SHOOTING QUERIES

Arc shooting is a generalization of the ray-shooting problem, where given a set S of objects, one wishes to find the first object of S hit by an oriented arc. Cheong *et al.* [59] have shown that a simple polygon P can be preprocessed into a data structure of linear size so that the first point of P hit by a circular or parabolic arc can be computed in $O(\log^2 n)$ time. Sharir and Shaul [118] have described a linear-size data structure that given a set of triangles in $\mathbb{R}^3$ can compute in $O(n^{3/4+\epsilon})$ time the first triangle hit by a vertical parabolic arc.

LINEAR-PROGRAMMING (LP) QUERIES

Let H be a set of n halfspaces in $\mathbb{R}^d$. *Preprocess H into a data structure so that for a direction vector u, the first point of $P(H) := \bigcap_{h \in H} h$ in the direction u can be determined quickly.* LP queries are generalizations of ray-shooting queries in the sense that we wish to compute the first point of $P(H)$ met by a hyperplane normal to u as we translate it in direction u.

For $d \leq 3$, an LP query can be answered in $O(\log n)$ time using $O(n)$ storage, by constructing the normal diagram of the convex polytope $P(S)$ and preprocessing it for point-location queries. For higher dimensions, Ramos [113] 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.

41.8 SOURCES AND RELATED MATERIAL

RELATED READING

Books and Monographs

[62]: Basic topics in computational geometry.

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

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

[93, 116]: Range-searching data structures in spatial database systems.

Survey Papers

[14, 97]: Range-searching data structures.

[75, 105, 115] Indexing techniques used in databases.

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

[25]: Secondary-memory data structures.

RELATED CHAPTERS

Chapter 30: Arrangements
Chapter 39: Point location
Chapter 42: Ray shooting and lines in space
Chapter 44: Nearest-neighbor searching in high dimensions
Chapter 54: Modeling motion

REFERENCES

[1] M. A. Abam, M. de Berg, and B. Speckmann. Kinetic kd-trees and longest-side kd-trees. *SIAM J. Comput.*, 39:1219–1232, 2009.

[2] P. Afshani. Improved pointer machine and I/O lower bounds for simplex range reporting and related problems. *Int. J. Comput. Geometry Appl.*, 23(4-5):233–252, 2013.

[3] P. Afshani, L. Arge, and K. D. Larsen. Orthogonal range reporting in three and higher dimensions. In *Proc. 50th Annual IEEE Sympos. Found. Comp. Sci.*, pages 149–158, 2009.

[4] P. Afshani, L. Arge, and K. D. Larsen. Orthogonal range reporting: query lower bounds, optimal structures in 3-d, and higher-dimensional improvements. In *Proc. 26th ACM Sympos. Comput. Geom.*, pages 240–246, 2010.

[5] P. Afshani, L. Arge, and K. G. Larsen. Higher-dimensional orthogonal range reporting and rectangle stabbing in the pointer machine model. In *Proc. 28th Annu. Sympos. Comput. Geom.*, pages 323–332, 2012.

[6] P. Afshani and T. M. Chan. Optimal halfspace range reporting in three dimensions. In *Proc. 20th Annu. ACM-SIAM Sympos. Discrete Algo.*, pages 180–186, 2009.

[7] P. Afshani, C. H. Hamilton, and N. Zeh. A general approach for cache-oblivious range reporting and approximate range counting. *Comput. Geom. Theory Appls.*, 43(8):700–712, 2010.

[8] P. Afshani and K. Tsakalidis. Optimal deterministic shallow cuttings for 3D dominance ranges. In *Proc. 25th Annu. ACM-SIAM Sympos. on Discrete Algo.*, pages 1389–1398, 2014.

[9] P. Afshani and N. Zeh. Improved space bounds for cache-oblivious range reporting. In *Proc. 22nd Annual ACM-SIAM Sympos. Discrete Algo.*, pages 1745–1758, 2011.

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

[11] 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, 1995.

[12] P. K. Agarwal, S. Cheng, and K. Yi. Range searching on uncertain data. *ACM Transactions on Algorithms*, 8:43, 2012.

[13] 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.

[14] 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.

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

[16] P. K. Agarwal, N. Kumar, S. Stavros, and S. Suri. Range-max queries on uncertain data. manuscript, 2016.

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

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

[19] P. K. Agarwal, J. Matousek, and M. Sharir. On range searching with semialgebraic sets. II. *SIAM J. Comput.*, 42:2039–2062, 2013.

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

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

[22] A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Commun. ACM*, 31:1116–1127, 1988.

[23] M. Ajtai. A lower bound for finding predecessors in Yao's call probe model. *Combinatorica*, 8(3):235–247, 1988.

[24] S. Alstrup, G. S. Brodal, and T. Rauhe. Optimal static range reporting in one dimension. In *Proc. 33rd Annu. ACM Sympos. Theory Comput.*, pages 476–482, 2001.

[25] 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.

[26] L. Arge, M. de Berg, H. J. Haverkort, and K. Yi. The priority R-tree: A practically efficient and worst-case optimal r-tree. In *Proc. ACM SIGMOD Intl. Conf. Manage. Data*, pages 347–358, 2004.

[27] 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.

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

[29] S. Arya and D. M. Mount. Approximate range searching. *Comput. Geom. Theo. Appls.*, 17(3-4):135–152, 2000.

[30] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923, 1998.

[31] S. Arya, D. M. Mount, and J. Xia. Tight lower bounds for halfspace range searching. *Discrete Comput. Geom.*, 47(4):711–730, 2012.

[32] A. Bagchi, A. Chaudhary, D. Eppstein, and M. T. Goodrich. Deterministic sampling and range counting in geometric data streams. *ACM Trans. Algo.*, 3, 2007.

[33] S. Barone and S. Basu. Refined bounds on the number of connected components of sign conditions on a variety. *Discr. Comput. Geom.*, 47:577–597, 2012.

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

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

[36] A. Beygelzimer, S. Kakade, and J. Langford. Cover trees for nearest neighbor. In *Proc. 23rd Intl. Conf. Machine Learning*, pages 97–104, 2006.

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

[38] G. S. Brodal and K. G. Larsen. Optimal planar orthogonal skyline counting queries. In *iProc. 14th Scand. Work. Algo. Theory*, pages 110–121, 2014.

[39] T. M. Chan. Optimal partition trees. *Discrete Comput. Geom.*, 47:661–690, 2012.

[40] T. M. Chan, S. Durocher, K. G. Larsen, J. Morrison, and B. T. Wilkinson. Linear-space data structures for range mode query in arrays. *Theory Comput. Syst.*, 55(4):719–741, 2014.

[41] T. M. Chan, K. G. Larsen, and M. Pătraşcu. Orthogonal range searching on the RAM, revisited. In *Proc. 27th Annu. Sympos. Comput. Geom.*, pages 1–10, 2011.

[42] T. M. Chan and M. Pătraşcu. Counting inversions, offline orthogonal range counting, and related problems. In *Proc. 21st Annu. ACM-SIAM Sympos. Discrete Algo.*, pages 161–173, 2010.

[43] T. M. Chan and K. Tsakalidis. Optimal deterministic algorithms for 2-d and 3-d shallow cuttings. In *Proc. 31st Intl. Sympos. Comput. Geom.*, pages 719–732, 2015.

[44] T. M. Chan and B. T. Wilkinson. Adaptive and approximate orthogonal range counting. In *Proc. 24th Annu. ACM-SIAM Sympos. Discrete Algo.*, pages 241–251, 2013.

[45] T. M. Chan and G. Zhou. Multidimensional range selection. In *Proc. 26th Intl. Sympos. Algo. Comput.*, pages 83–92, 2015.

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

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

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

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

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

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

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

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

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

[55] B. Chazelle, L. J. Guibas, and D. T. Lee. The power of geometric duality. *BIT*, 25(1):76–90, 1985.

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

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

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

[59] S. Cheng, O. Cheong, H. Everett, and R. van Oostrum. Hierarchical decompositions and circular ray shooting in simple polygons. *Discrete Comput. Geom.*, 32:401–415, 2004.

[60] S. Dasgupta and Y. Freund. Random projection trees and low dimensional manifolds. In *Proc. 40th Annu. ACM Sympos. Theory Comput.*, pages 537–546, 2008.

[61] S. Dasgupta and K. Sinha. Randomized partition trees for nearest neighbor search. *Algorithmica*, 72:237–263, 2015.

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

[63] P. F. Dietz and R. Raman. Persistence, amortization and randomization. In *Proc. 2nd Annu. ACM-SIAM Sympos. Discrete Algo.*, pages 78–88. 1991.

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

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

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

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

[68] 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.

[69] D. Felber and R. Ostrovsky. A randomized online quantile summary in $o(1/\epsilon * \log(1/\epsilon))$ words. In *Proc. APPROX/RANDOM*, pages 775–785, 2015.

[70] J. Fischer and V. Heun. Theoretical and practical improvements on the RMQ-problem, with applications to LCA and LCE. In *Proc. 17th Anu. Sympos. Combinatorial Pattern Matching*, pages 36–48, 2006.

[71] 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.

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

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

[74] M. L. Fredman. The complexity of maintaining an array and computing its partial sums. *J. ACM*, 29:250–260, 1982.

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

[76] 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.

[77] 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.

[78] M. Greenwald and S. Khanna. Space-efficient online computation of quantile summaries. In *Proc. ACM SIGMOD Intl. Conf. Manage. Data*, pages 58–66, 2001.

[79] L. Guth and N. H. Katz. On the Erdős distinct distances problem in the plane. *Annals Math.*, 181:155–190, 2015.

[80] A. Guttman. R-trees: A dynamic index structure for spatial searching. In *Proc. 3rd Annu. ACM Sympos. Principles Database Systems*, pages 47–57, 1984.

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

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

[83] C.-T. Ho, J. Bruck, and R. Agrawal. Partial-sum queries in OLAP data cubes using covering codes. In *Proc. 16th Annu. ACM Sympos. Principles Database Syst.*, pages 228–237, 1997.

[84] J. JáJá, C. W. Mortensen, and Q. Shi. Space-efficient and fast algorithms for multidimensional dominance reporting and counting. In *Proc. 15th Intl. Sympos. Algo. Comput.*, pages 558–568, 2004.

[85] R. Janardan and M. Lopez. Generalized intersection searching problems. *Internat. J. Comput. Geom. Appl.*, 3:39–69, 1993.

[86] A. G. Jørgensen and K. G. Larsen. Range selection and median: Tight cell probe lower bounds and adaptive data structures. In *Proc. 22nd Annual ACM-SIAM Sympos. Discrete Algo.*, pages 805–813, 2011.

[87] H. Kaplan, N. Rubin, M. Sharir, and E. Verbin. Efficient colored orthogonal range counting. *SIAM J. Comput.*, 38:982–1011, 2008.

[88] K. G. Larsen. The cell probe complexity of dynamic range counting. In *Proc. 44th Annu. Sympos. Theory Comput.*, pages 85–94, 2012.

[89] K. G. Larsen. On range searching in the group model and combinatorial discrepancy. *SIAM J. Comput.*, 43:673–686, 2014.

[90] K. G. Larsen and H. L. Nguyen. Improved range searching lower bounds. In *Proc. 28th Annu. Sympos. Comput. Geom.*, pages 171–178, 2012.

[91] K. G. Larsen and F. van Walderveen. Near-optimal range reporting structures for categorical data. In *Proc. 24th Annual ACM-SIAM Sympos. Discrete Algo.*, pages 265–276, 2013.

[92] 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.

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

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

[95] J. Matoušek. Reporting points in halfspaces. *Comput. Geom. Theory Appl.*, 2(3):169–186, 1992.

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

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

[98] J. Matousek and Z. Patáková. Multilevel polynomial partitions and simplified range searching. *Discrete Comput. Geom.*, 54:22–41, 2015.

[99] J. Matousek and O. Schwarzkopf. Linear optimization queries. In *Proc. 8th Annu. Sympos. Comput. Geom.*, pages 16–25, 1992.

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

[101] C. W. Mortensen, R. Pagh, and M. Pătraşcu. On dynamic range reporting in one dimension. In *Proc. 37th Annual ACM Sympos. Theory Comput.*, pages 104–111, 2005.

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

[103] S. Muthukrishnan. Data streams: Algorithms and applications. *Foundations and Trends in Theoretical Computer Science*, 1, 2006.

[104] Y. Nekrich. Efficient range searching for categorical and plain data. *ACM Trans. Database Syst.*, 39:9, 2014.

[105] 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.

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

[107] 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.

[108] M. Pătraşcu. Lower bounds for 2-dimensional range counting. In *Proc. 39th Annual ACM Sympos. Theory Comput.*, pages 40–46, 2007.

[109] M. Pătraşcu. Unifying the landscape of cell-probe lower bounds. *SIAM J. Comput.*, 40:827–847, 2011.

[110] M. Pătraşcu and E. D. Demaine. Logarithmic lower bounds in the cell-probe model. *SIAM J. Comput.*, 35:932–963, 2006.

[111] S. Rahul. Approximate range counting revisited. *CoRR*, abs/1512.01713, 2015.

[112] S. Rahul and R. Janardan. Algorithms for range-skyline queries. In *Proc. Intl. Conf. Adv. Geog. Inf. Sys.*, pages 526–529, 2012.

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

[114] J. T. Robinson. The k-d-B-tree: A search structure for large multidimensional dynamic indexes. Report CMU-CS-81-106, Dept. Comput. Sci., Carnegie-Mellon Univ., Pittsburgh, PA, 1981.

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

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

[117] 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.

[118] M. Sharir and H. Shaul. Ray shooting and stone throwing with near-linear storage. *Comput. Geom. Thery Appls.*, 30:239–252, 2005.

[119] M. Streppel and K. Yi. Approximate range searching in external memory. *Algorithmica*, 59(2):115–128, 2011.

[120] S. Suri, C. D. Tóth, and Y. Zhou. Range counting over multidimensional data streams. In *Proc. 20th Sympos. Comput. Geom.*, pages 160–169, 2004.

[121] P. van Emde Boas, R. Kaas, and E. Zijlstra. Design and implementation of an efficient priority queue. *Mathematical Systems Theory*, 10:99–127, 1977.

[122] S. Vempala. Randomly-oriented k-d trees adapt to intrinsic dimension. In *Proc. IARCS Annu. Conf. Found. Soft. Tech. Theo. Comp. Sci.*, pages 48–57, 2012.

[123] J. S. Vitter and M. Wang. Approximate computation of multidimensional aggregates of sparse data using wavelets. In *Proc. ACM SIGMOD Intl. Conf. Management Data*, pages 193–204, 1999.

[124] 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.

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

[126] Z. Wei and K. Yi. The space complexity of 2-dimensional approximate range counting. In *Proc. 24th Annual ACM-SIAM Sympos. Discrete Algo.*, pages 252–264, 2013.

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

[128] A. C. Yao. Should tables be sorted? *J. ACM*, 28(3):615–628, 1981.

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

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