Simplex Range Searching and Its Variants: A Review*

Pankaj K. Agarwal
Department of Computer Science
Duke University
Durham, NC 27708-0129

February 28, 2016

Abstract

A central problem in computational geometry, range searching arises in many applications, and numerous geometric problems can be formulated in terms of range searching. A typical range-searching problem has the following form. Let S be a set of n points in $\mathbb{R}^d$, and let $\mathcal{R}$ be a family of subsets of $\mathbb{R}^d$; elements of $\mathcal{R}$ are called *ranges*. Preprocess S into a data structure so that for a query range $\gamma \in \mathcal{R}$, the points in $S \cap \gamma$ can be reported or counted efficiently. Notwithstanding extensive work on range searching over the last four decades, it remains an active research area. A series of papers by Jirka Matoušek and others in the late 1980s and the early 1990s had a profound impact not only on range searching but also on computational geometry as a whole. This chapter reviews the known results and techniques, including recent developments, for simplex range searching and its variants.

*Work on this chapter is supported by NSF under grants CCF-11-61359, IIS-14-08846, and CCF-15-13816, by an ARO grant W911NF-15-1-0408, and by Grant 2012/229 from the U.S.-Israel Binational Science Foundation.

1 Introduction

In the mid 1980s, the range-searching problem, especially simplex range searching, was wide open: neither efficient algorithms nor nontrivial lower bounds were known. A series of papers in the late 1980s and the early 1990s [37, 38, 58, 69, 70, 82] not only marked the beginning of a new chapter in range searching but also revitalized computational geometry as a whole. The impact of techniques developed for range searching — ε-nets, $(1/r)$-cuttings, partition trees, simplicial partitions, multi-level data structures, to name a few — is evident throughout computational geometry. The papers by Jirka Matoušek [67, 68, 69, 70, 71, 72] were at the center of this range-searching revolution. This book honoring his work provides an excellent opportunity to review the current status of geometric range searching and to summarize the recent progress in this area.

A typical range-searching problem has the following form: Let S be a set of n points in $\mathbb{R}^d$, and let $\mathcal{R}$ be a family of subsets of $\mathbb{R}^d$; elements of $\mathcal{R}$ are called *ranges*. The goal is to preprocess S into a data structure so that for a query range $\gamma \in \mathcal{R}$, the points in $S \cap \gamma$ can be reported or counted efficiently. Typical examples of ranges include rectangles, halfspaces, simplices, and balls. 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 applications, however, call for querying the same point set S several 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 queries can be encompassed in the following general formulation of range searching.

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 $\gamma \in \mathcal{R}$, the goal is to compute $w(S \cap \gamma)$. For example, counting queries can be answered by choosing the semigroup to be $(\mathbb{Z}, +)$, where $+$ denotes standard integer addition, and setting $w(p) = 1$ for every $p \in S$; emptiness queries by choosing the semigroup to be $(\{0, 1\}, \vee)$ and setting $w(p) = 1$; reporting queries by choosing the semigroup to be $(2^S, \cup)$ and setting $w(p) = \{p\}$; and range-max queries by choosing the semigroup to be $(\mathbb{R}, \max)$ and choosing $w(p)$ to be the weight of p.

The performance of a data structure is measured by the time spent in answering a query, called the *query time*; by the *size* of the data structure; and by the time spent in constructing the data structure, called the *preprocessing time*. Since the data structure is constructed only once, its query time and size are generally more important than its preprocessing time. We should remark that the query time of a range-reporting query on any reasonable machine depends on the output size, so the query time for a range-reporting query consists of two parts — *search time*, which depends only on n and d; and *reporting time*, which depends on n, d, and the output size. Throughout this chapter we will use k to denote the output size.

We assume that d is a small fixed constant, and that big-O and big-Ω notation hide constants depending on d. The size of any range-searching data structure is at least linear, since it has to store each point (or its weight) at least once, and the query time in any reasonable model of computation such as pointer machine, RAM, or algebraic decision tree is $\Omega(\log n)$ even for $d = 1$, assuming the coordinates of input points are real values. Ideally, one would like to develop a

linear-size data structure with logarithmic query time. If such a data structure is not feasible, then one seeks a tradeoff between the query time and the size of the data structure — How fast can a query be answered using $O(n \operatorname{polylog} n)$ space, how much space is required to answer a query in $O(\operatorname{polylog} n)$ time, and what kind of tradeoff between the size and the query time can be achieved?

The early work on range searching focused on *orthogonal range searching*, where ranges are axis-parallel boxes [21, 22]. Even after four decades of extensive work on orthogonal range searching, some basic questions still remain open; see the survey papers [4, 5]. Geometry plays almost no role in the known data structures for orthogonal range searching. The most basic and most studied truly geometric instance of range searching is with *halfspaces*, or more generally *simplices*, as ranges. We therefore focus on simplex range searching and its variants.

The chapter is organized as follows. We describe, in Section 2, different models of computation that have been used to prove upper and lower bounds on the performance of range-searching data structures. Next, Section 3 surveys known techniques and data structures for simplex range searching. Section 4 focuses on the special case of halfspace range reporting for which faster data structures are known. Section 5 reviews semialgebraic range searching, where there has been some recent progress using algebraic techniques. Finally, Section 6 discusses a few variants and extensions of range searching.

2 Models of Computation

Most algorithms and data structures in computational geometry are implicitly described in the familiar *random access machine* (RAM) model or the *real RAM* model. In the traditional RAM model, memory cells can contain arbitrary $(\log n)$-bit integers, which can be added, multiplied, subtracted, divided (computing $\lfloor x/y \rfloor$), compared, and used as pointers to other memory cells in constant time. In a real RAM, memory cells can store arbitrary real numbers (such as coordinates of points), and basic arithmetic and relational operations between real numbers can be performed in constant time. In the case of range searching over a semigroup other than the integers, memory cells are allowed to contain arbitrary values from the semigroup, but these values can only be added (using the semigroup's addition operator, of course).

All range-searching data structures discussed in this chapter can be described in the more restrictive *pointer machine* model. The main difference between the pointer-machine and the RAM 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 a portion of the graph and visits at least one node with label i for each point p_i in the query range. Chazelle [27] defines 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.

Most lower bounds, and a few upper bounds, are described in the so-called *semigroup arithmetic model*, which was originally introduced by Fredman [54] and refined by Yao [84]. In the semigroup arithmetic model, a data structure can be *informally* regarded as a set of precomputed partial sums in the underlying semigroup. The size of the data structure is the number of sums stored, and the

query time is the minimum number of semigroup operations required (on the precomputed sums) to compute the answer to a query. The query time ignores the cost of various auxiliary operations, including the cost of determining which of the precomputed sums should be added to answer a query. Unlike the pointer-machine model, the semigroup model allows immediate access, at no cost, to any precomputed sum. The informal model we have just described is much too powerful. For example, in this informal model, the optimal data structure for counting queries consists of the $n + 1$ integers $0, 1, \ldots, n$. To answer a counting query, we simply return the correct answer; since no additions are required, we can answer queries in zero "time", using a "data structure" of only linear size! A more formal definition, using the notion of a *faithful semigroup*, that avoids this problem can be found in [28].

A 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, we will also consider the *group model*, 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. 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 [31]. Unfortunately, no such property holds for the group model, which makes proving lower bounds in the group model much harder. The known lower bounds for range searching in the group model are much weaker than those under the semigroup model.

Many 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 [51, 52], 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, they precompute a family $\mathcal{F} = \mathcal{F}(S)$ of *canonical subsets* of S and store the weight $w(C) = \sum_{p \in C} w(p)$ of each canonical subset $C \in \mathcal{F}$. For a query range γ, they determine a partition $\mathcal{C}_\gamma = \mathcal{C}(S, \gamma) \subseteq \mathcal{F}$ of $S \cap \gamma$ and add the weights of the subsets in $\mathcal{C}_\gamma$ to compute $w(S \cap \gamma)$. Borrowing terminology from [72], we refer to such a data structure as a *decomposition scheme*. There is a close connection between decomposition schemes and partial sums stored in the semigroup arithmetic model described earlier—$w(C)$, the weight of each canonical subset C, corresponds to a precomputed partial sum.

How exactly the weights of canonical subsets are stored and how $\mathcal{C}_\gamma$ is computed depends on the model of computation and on the specific range-searching problem. In the semigroup (or group) arithmetic model, the query time depends only on the number of canonical subsets in $\mathcal{C}_\gamma$, regardless of how they are computed, so the weights of canonical subsets can be stored in an arbitrary manner. In more realistic models of computation, however, some additional structure must be imposed on the decomposition scheme in order to efficiently compute $\mathcal{C}_\gamma$. In a *hierarchical* decomposition scheme, the weights are stored in a tree T. Each node v of T is associated with a

canonical subset $C_v \in \mathcal{F}$, and the children of v are associated with subsets of C_v. Besides the weight of C_v, some auxiliary information is also stored at v, which is used to determine whether $C_v \in \mathcal{C}_\gamma$ for a query range γ. Typically, this auxiliary information consists of some geometric object, which plays the same role as a query region in the partition graph model.

If the weight of each canonical subset can be stored in $O(1)$ memory cells, then the total size of the data structure is just $O(|\mathcal{F}|)$. If the underlying searching problem is a range-reporting problem, however, then the "weight" of a canonical subset is the set itself, and thus it is not realistic to assume that each "weight" requires only constant space. In this case, the size of the data structure is $O(\sum_{C \in \mathcal{F}} |C|)$ if each subset is stored explicitly at each node of the tree. However, the size can be reduced to $O(|\mathcal{F}|)$ by storing the subsets implicitly (*e.g.*, storing points only at leaves).

Finally, let $r \geq 2$ be a parameter, and set $\mathcal{F}_i = \{C \in \mathcal{F} \mid r^{i-1} \leq |C| \leq r^i\}$. A hierarchical decomposition scheme is called *r-convergent* if there exist constants $\alpha \geq 1$ and $\beta \geq 0$ so that the degree of every node in T is $O(r^\alpha)$ and for all $i \geq 1$, $|\mathcal{F}_i| = O((n/r^i)^\alpha)$ and, for all query ranges γ, $|\mathcal{C}_\gamma \cap \mathcal{F}_i| = O((n/r^i)^\beta)$, i.e., the number of canonical subsets in the data structure and in any query output decreases exponentially with their size. The size of the decomposition scheme is $O(n^\alpha)$, provided the weight of each canonical subset can be stored in $O(1)$ space.

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 of T, starting from the root. At each node v, using the auxiliary information stored at v, the procedure determines whether the query range γ contains C_v, intersects C_v, or is disjoint from C_v. If γ contains C_v, then C_v is added to $\mathcal{C}_\gamma$ (rather, the weight of C_v is added to a running counter). Otherwise, if γ intersects C_v, the query procedure identifies a subset of children of v, say $\{w_1, \ldots, w_a\}$, so that the canonical subsets $C_{w_i} \cap \gamma$, for $1 \leq i \leq a$, form a partition of $C_v \cap \gamma$. Then the procedure searches each w_i recursively. If the decomposition scheme is r-convergent, then its query time, under the semigroup model, is $O(n^\beta)$ if $\beta > 0$ and $O(\log n)$ if $\beta = 0$. A decomposition scheme is called *efficient* if for any query range γ, each $\mathcal{C}_\gamma \cap \mathcal{F}_i$ can be computed in time $O\left((n/r^i)^\beta\right)$.

We will see below in Section 6 that r-convergent hierarchical decomposition schemes can be cascaded together to construct multi-level structures that answer complex geometric queries.

3 Simplex Range Searching

In this section we focus on simplex range searching, the case in which the query ranges are simplices. No data structure is known that can answer a simplex range 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 saving only a factor of $n^{1/d}$ over the naïve approach). Since the size and query time of any data structure have to be at least linear and logarithmic, respectively, we consider these two ends of the spectrum: (i) How large should the size of a data structure be in order to answer a query in logarithmic time, and (ii) how fast can a simplex range query be answered using a linear-size data structure. By combining these two extreme cases, as we describe below, a tradeoff between space and query time can be obtained.

3.1 Data structures with logarithmic query time

A common approach to answer a range query in $O(\log n)$ time is using the *locus* approach that, roughly speaking, works as follows: Let S be a set of weighted points in $\mathbb{R}^d$ and $\mathcal{R}$ a family of ranges (e.g. the set of all halfspaces, or the set of all simplices). If each range $\gamma \in \mathcal{R}$ is specified by b real numbers, then γ can be mapped to a point γ^* in a b-dimensional space, which we denote by $\mathcal{R}^*$. Each input point $p \in S$ is mapped to a region $p^* \subseteq \mathcal{R}^*$ such that $p \in \gamma$ if and only if $\gamma^* \in p^*$. Set $S^* = \{p^* \mid p \in S\}$. $\mathcal{R}^*$ is partitioned into connected "cells" so that all points within each cell τ lie in the same subset S^*_τ of S^*. We store $w_\tau = \sum_{p^* \in S^*_\tau} w(p)$ for each cell τ of the subdivision. A range query with $\gamma \in \mathcal{R}$ then reduces to locating the point γ^* in this subdivision of $\mathcal{R}^*$, identifying the cell τ that contains γ^*, and returning w_τ.

For the sake of simplicity, we first illustrate this approach for the halfspace range-counting problem. We need a few definitions and concepts before we describe the data structures.

The *dual* of a point $p = (a_1, \ldots, a_d) \in \mathbb{R}^d$ is the hyperplane $p^* : x_d = a_1 x_1 + \cdots + a_{d-1} x_{d-1} - a_d$, and the dual of a hyperplane $h : x_d = b_1 x_1 + \cdots + b_{d-1} x_{d-1} + b_d$ is the point $h^* = (b_1, \ldots, b_{d-1}, -b_d)$. A nice property of duality is that a point p is above (resp. on below) a hyperplane h if and only if the dual point h^* is above (resp. on below) the dual hyperplane p^*.

The *arrangement* of a set H of hyperplanes in $\mathbb{R}^d$, denoted by $\mathcal{A}(H)$, is the subdivision of $\mathbb{R}^d$ into cells of dimensions k, for $0 \leq k \leq d$, each cell being a maximal connected set contained in the intersection of a fixed subset of H and not intersecting any other hyperplane of H. The *level* of a point in $\mathcal{A}(H)$ is the number of hyperplanes lying strictly below the point. Let $\mathcal{A}_{\leq k}(H)$ denote the (closure of the) set of points with level at most k.

For a parameter $r \in [1, n]$, 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. Clarkson [40] and Haussler and Welzl [58] were the first to show the existence of a $(1/r)$-cutting of H of size $O(r^d \log^d r)$. Chazelle and Friedman [32] improved the size bound to $O(r^d)$, which is optimal in the worst case. Matoušek [68] was the first to develop an efficient algorithm for constructing a $(1/r)$-cutting. The best algorithm known for computing a $(1/r)$-cutting was discovered by Chazelle [29]; his result is summarized in the following theorem.

Theorem 3.1 (Chazelle [29]). *Let H be a set of n hyperplanes in $\mathbb{R}^d$, let $r \leq n$ be a parameter, and let $b > 1$ be a constant. Set $s = \lceil \log_b r \rceil$. There exist s cuttings $\Xi_1, \ldots, \Xi_s$ so that Ξ_i is a $(1/b^i)$-cutting of size $O(b^{id})$, each simplex of Ξ_i is contained in a simplex of Ξ_{i-1}, and each simplex of Ξ_{i-1} contains a constant number of simplices of Ξ_i. Moreover, $\Xi_1, \ldots, \Xi_s$ can be computed in time $O(nr^{d-1})$.*

The key idea of Chazelle is that Ξ_i is constructed by refining each simplex $\triangle \in \Xi_{i-1}$. Let $H_\triangle \subseteq H$ be the set of hyperplanes that cross $\triangle$, let $n_\triangle = |H_\triangle|$, and let $\chi_\triangle$ be the number of vertices of $\mathcal{A}(H)$ that lie in the interior of $\triangle$. Chazelle's algorithm computes a $(1/b)$-cutting of $H_\triangle$ within $\triangle$ whose size depends on $\chi_\triangle$. More precisely, it partitions $\triangle$, in time $O(n_\triangle b^{O(1)})$, into a set $\Xi_\triangle$ of simplices so that each of them is crossed by at most $n_\triangle / b \leq n/b^i$ hyperplanes of H, and more importantly $|\Xi_\triangle| = O(\chi_\triangle (\frac{b^i}{n})^d + b^{d-1})$.

Returning to halfspace range searching, suppose that the query halfspace always lies below its bounding hyperplane. Then the halfspace range-counting problem reduces via duality to the following problem: Given a set H of n hyperplanes in $\mathbb{R}^d$, determine the number of hyperplanes of H that lie above a query point. Theorem 3.1 can be used in a straightforward manner to obtain a data structure of size $O(n^d / \log^{d-1} n)$ that can return, in $O(\log n)$ time, the number of hyperplanes

of H lying above a query point ξ as follows: 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 $\xi \in \mathbb{R}^d$, by traversing the pointers, find the simplex $\triangle \in \Xi_s$ that contains ξ, count the number of hyperplanes of $H_\triangle$ that lie above ξ, and return $k_\triangle$ plus this quantity. The total query time is $O(\log n)$.

The above locus approach for halfspace range counting can be extended to the simplex range-counting problem as well. That is, we map a d-simplex $\triangle$ to a point $\triangle^* \in \mathbb{R}^{d(d+1)}$ and each point $p \in S$ to a region $p^* \subset \mathbb{R}^{d(d+1)}$ such that $p \in \triangle$ if and only if $\triangle^* \in p^*$. Then a simplex range-counting query reduces to point location in the arrangement $\mathcal{A}(S^*)$. Since $\mathcal{A}(S^*)$ has $\Omega(n^{d(d+1)})$ cells, such an approach will require $\Omega(n^{d(d+1)})$ storage; see [43, 48]. More efficient data structures have been developed using the multi-level decomposition scheme mentioned in Section 2 and described in Section 6.1.

Cole and Yap [43] were the first to present a near-quadratic size data structure that could answer a triangle range-counting query in the plane in $O(\log n)$ time. They present two data structures: the first one answers a query in time $O(\log n)$ using $O(n^{2+\varepsilon})$ space, and the other in time $O(\log n \log\log n)$ using $O(n^2/\log n)$ space. For $d = 3$, their approach gives a data structure of size $O(n^{7+\varepsilon})$ that can answer a tetrahedron range-counting query in time $O(\log n)$. Chazelle *et al.* [37] describe a multi-level data structure of size $O(n^{d+\varepsilon})$ that can answer a simplex range-counting query in time $O(\log n)$. The space bound can be reduced to $O(n^d)$ by increasing the query time to $O(\log^{d+1} n)$ [72]. These data structures can answer simplex range-reporting queries by spending an additional $O(k)$ time.

3.2 Linear-size data structures

Most of the linear-size data structures for simplex range searching are based on *partition trees*, originally introduced by Willard [83]. Roughly speaking, partition trees are based on the following idea: Given a set S of points in $\mathbb{R}^d$, partition the space into a few, say, a constant number of, regions, each containing roughly equal number of points, so that for any hyperplane h, the number of points lying in the regions that intersect h is much less than the total number of points. Then recursively construct a similar partition for the subset of points lying in each region.

Willard's original partition tree for a set S of n points in the plane is a 4-way tree, constructed as follows. Let us assume that n is of the form 4^k for some integer k, and that the points of S are in general position. If $k = 0$, the tree consists of a single node that stores the coordinates of the only point in S. Otherwise, using the ham-sandwich theorem [76], find two lines ℓ_1, ℓ_2 so that each quadrant Q_i, for $1 \leq i \leq 4$, induced by ℓ_1, ℓ_2 contains exactly $n/4$ points. The root stores the equations of ℓ_1, ℓ_2 and the value of n. For each quadrant, recursively construct a partition tree for $S \cap Q_i$ and attach it as the i^{th} subtree of the root. The total size of the data structure is linear, and it can be constructed in $O(n \log n)$ time. A halfplane range-counting query can be answered by refining the generic procedure described in Section 2, as follows: Let h be a query halfplane. Traverse the tree, starting from the root, and maintain a global count. At each node v, perform the following step: If the line ∂h intersects the quadrant Q_v associated with v, recursively visit the children of v. If $Q_v \cap h = \emptyset$, do nothing. Otherwise, since $Q_v \subseteq h$, add the number of points of S lying in the subtree rooted at S_v to the global count. The quadrants associated with the four

children of any interior node are induced by two lines, so ∂h intersects at most three of them, which implies that the query procedure does not explore the subtree of one of the children. Hence, the query time of this procedure is $O(n^\alpha)$, where $\alpha = \log_3 4 \leq 0.7925$. A similar procedure can answer a triangle range-counting query within the same time bound, and a triangle range-reporting query in time $O(n^\alpha + k)$. Edelsbrunner and Welzl [50] described a simple variant of Willard's partition tree that improved the exponent in the query-search time to $\log_2(1+\sqrt{5}) - 1 \approx 0.695$.

A partition tree for points in $\mathbb{R}^3$ was first proposed by Yao [86], which can answer a counting query in time $O(n^{0.98})$. Using the Borsuk-Ulam theorem (see the monograph by Matoušek [76]), Yao *et al.* [87] showed that, given a set S of n points in $\mathbb{R}^3$, one can find three planes so that each of the eight (open) octants determined by them contains at most $\lfloor n/8 \rfloor$ points of S. This approach leads to a linear-size data structure with query time $O(n^{0.899})$. Avis [19] proved that such a partition of $\mathbb{R}^d$ by d hyperplanes is not always possible for $d \geq 5$; the problem is still open for $d = 4$. Weaker partitioning schemes for $d \geq 4$ were proposed in [42, 85].

After the initial improvements and extensions on Willard's partition tree, a major breakthrough was made by Haussler and Welzl [58]. 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 $\alpha = 1 - \frac{1}{d(d-1)+1} + \varepsilon$ for any $\varepsilon > 0$. Besides an improved range searching data structure, the major contribution of their paper is the abstract framework and the notion of ε-nets. This and related abstract frameworks have popularized randomized algorithms in computational geometry [45].

The first linear-size data structure with near-optimal query time for simplex range queries in the plane was developed by Welzl [82]. His algorithm is based on the following idea. A *spanning path* of a set S of points is a polygonal chain whose vertices are the points of S. The *crossing number* of a polygonal path is the maximum number of its edges that can be crossed by a hyperplane. Welzl constructs a spanning path $\Pi = \Pi(S)$ of any set S of n points in $\mathbb{R}^d$ whose crossing number is $O(n^{1-1/d} \log n)$. The bound on the crossing number was improved by Chazelle and Welzl [38] to $O(n^{1-1/d})$, which is tight in the worst case. Let $p_1, p_2, \ldots, p_n$ be the vertices of Π. If we know the edges of Π that cross h, then the weight of points lying in one of the halfspaces bounded by h can be computed by answering $O(n^{1-1/d})$ partial-sum queries on the sequence $W = \langle w(p_1), \ldots, w(p_n) \rangle$. Hence, by processing W for partial-sum queries, we obtain a linear-size data structure for simplex range searching, with $O(n^{1-1/d}\alpha(n))$ query time, in the semigroup arithmetic model, where $\alpha(n)$ is the inverse Ackermann function. (Recall that the time spent in finding the edges of Π crossed by h is not counted in the semigroup model.) In any realistic model of computation such as pointer machines or RAMs, however, we also need an efficient linear-size data structure for computing the edges of Π crossed by a hyperplane. Chazelle and Welzl [38] produced such a data structure for $d \leq 3$, but no such structure is known for higher dimensions.

The first data structure with roughly $n^{1-1/d}$ query time and near-linear space, for $d > 3$, was obtained by Chazelle *et al.* [37]. Given a set S of n points in $\mathbb{R}^d$, they construct a family $\mathcal{F} = \{\Xi_1, \ldots, \Xi_k\}$ of triangulations of $\mathbb{R}^d$, each of size $O(r^d)$. For any hyperplane h, there is at least one Ξ_i so that only $O(n/r)$ points lie in the simplices of Ξ_i that intersect h. Applying this construction recursively, they obtain a tree structure of size $O(n^{1+\varepsilon})$ that can answer a halfspace range-counting query in time $O(n^{1-1/d})$. The extra n^ε factor in the space is due to the fact that they maintain a family of partitions instead of a single partition. Another consequence of maintaining a family of partitions is that, unlike partition trees, this data structure cannot be used directly to answer simplex range queries. Instead, Chazelle *et al.* [37] construct a multi-level data structure

(Section 6.1) to answer simplex range queries.

Matoušek [72] developed a simpler, slightly faster data structure for simplex range queries, by returning to the theme of constructing a single partition, as in the earlier partition-tree papers. His algorithm is based on the following partition theorem, which can be regarded as an extension of the results by Welzl [82] and Chazelle and Welzl [38].

Theorem 3.2 (Matoušek [69]). *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 each $S_i \subseteq S$ lies inside the simplex Δ_i, $n/r \le |S_i| \le 2n/r$, $S_i \cap S_j = \emptyset$ for all $i \ne j$, and every hyperplane crosses at most $cr^{1-1/d}$ simplices of Π; here c is a constant. If $r \le n^\alpha$ for some suitable constant $0 < \alpha < 1$, then Π can be constructed in $O(n \log r)$ time.*

Note that although S is being partitioned into a family of subsets, unlike the earlier results on partition trees, it does not partition $\mathbb{R}^d$ because Δ_i's may intersect. Theorem 3.2 is proved by constructing a "test set" Γ of $O(r^d)$ hyperplanes so that if each hyperplane in Γ crosses $O(r^{1-1/d})$ simplices in Π, then the same holds for an arbitrary hyperplane [69]. Given Γ and S, the pairs in Π are constructed one by one using an iterative reweighing scheme that assigns weights to the hyperplanes in Γ and computes a (weighted) $(1/cr^d)$-cutting of Γ at each stage, analogous to the construction of the spanning path in [38, 82]; see [69] for details.

Using Theorem 3.2, a partition tree T can be constructed as follows. Each interior node v of T is associated with a canonical subset $C_v \subseteq S$ and a simplex Δ_v containing C_v; if v is the root of T, then $C_v = S$ and $\Delta_v = \mathbb{R}^d$. 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, v stores Δ_v and $|C_v|$, and we construct a family of pairs $\Pi_v = \{(S_1, \Delta_1), \ldots, (S_m, \Delta_m)\}$ using Theorem 3.2. We recursively construct a partition tree T_i for each S_i and attach T_i as the ith subtree of v. The total size of the data structure is linear, and it can be constructed in time $O(n \log n)$. A simplex range-counting query can be answered in the same way as with Willard's partition tree. Since any hyperplane intersects at most $cr^{1-1/d}$ simplices of Π, the query time is $O(n^{1-1/d+\log_r c})$; the $\log_r c$ term in the exponent can be reduced to any arbitrarily small positive constant ε by choosing r sufficiently large. The query time can be improved to $O(n^{1-1/d} \operatorname{polylog} n)$ by choosing $r = n^\varepsilon$.

In a subsequent paper, Matoušek [72] proved a stronger version of Theorem 3.2, using some additional sophisticated techniques (including Theorem 3.1), that gives a linear-size partition tree with $O(n^{1-1/d})$ query time. His scheme was subsequently simplified by Chan [25]. Unlike the recursive construction of the partition tree described above, Chan's algorithm does not construct the subtree at each node of the partition tree independently. Instead, roughly speaking, it constructs the partition tree level by level, where each level corresponds to a triangulation that "covers" all points of S (i.e., each point of S lie in a simplex of the triangulation), and each step partitions one of the simplices of the previous level into subsimplices. Therefore the triangulation at level $i+1$ is a refinement of the triangulation at level i, and the entire tree corresponds to a hierarchical triangulation in $\mathbb{R}^d$. The main result in [25] is the following theorem:

Theorem 3.3 (Chan [25]). *Let S be a set of n points in $\mathbb{R}^d$, H a set of m hyperplanes in $\mathbb{R}^d$, and $b \ge 1$ a parameter. Given t disjoint simplices that cover P such that each simplex contains at most $2n/t$ points of S and each hyperplane of H crosses at most ℓ cells, each simplex can be partitioned into $O(b)$ (disjoint) subsimplices, for a total of $O(bt)$ simplices such that each subsimplex contains at most $\frac{2n}{bt}$ points of S and each hyperplane of H crosses $O((bt)^{1-1/d} + b^{1-1/(d-1)}\ell + b \log t \log m)$ subsimplices.*

Note that the first term is the same as in Theorem 3.2. It is crucial that the coefficient $b^{1-1/(d-1)}$ in the second term is asymptotically smaller than $b^{1-1/d}$, for this ensures that the repeated application of this theorem does not incur a constant-factor blow up, as in the recursive construction based on Theorem 3.2. The partition tree based on Theorem 3.3 can be constructed in $O(n \log n)$ randomized expected time. An interesting by product of Chan's technique is that it can be used to construct a triangulation of S so that any hyperplane crosses the interior of $O(n^{1-1/d})$ simplices, thereby solving a long-standing open problem.

If the points in S lie on a k-dimensional algebraic surface of constant degree, the *crossing number* in Theorem 3.2 can be improved to $O(r^{1-1/\gamma})$, where $\gamma = 1/\lfloor (d+k)/2 \rfloor$ [7], which implies that in this case a simplex range query can be answered in time $O(n^{1-1/\gamma+\varepsilon})$ using linear space.

3.3 Trading space for query time

In the previous two subsections we surveyed data structures for simplex range searching that either use near-linear space or answer a query in polylogarithmic time. By combining these two types of data structures, a tradeoff between the size and the query time can be obtained [9, 25, 37, 72]. Actually, the approach described in these papers is very general and works for any geometric-searching data structure that can be viewed as a hierarchical decomposition scheme (described in Section 2), provided it satisfies certain assumptions. We state the general result here, though a slightly better bounds (by a polylogarithmic factor) can be obtained by exploiting special properties of the data structures.

Theorem 3.4. *Let S be a set of n points in $\mathbb{R}^d$, and let r be a sufficiently large constant. Let $\mathcal{P}$ be a range-searching problem. Let D^1 be a decomposition scheme for $\mathcal{P}$ of size $O(n^\alpha)$ and query time $O(\log n)$, and let $\mathcal{D}^2$ be another decomposition scheme of size $O(n)$ and query time $O(n^\beta)$. If either $\mathcal{D}^1$ or $\mathcal{D}^2$ is hierarchical, efficient, and r-convergent, then for any $n \leq m \leq n^\alpha$, then a decomposition scheme for $\mathcal{P}$ of size $O(m)$ can be constructed that has $O((\frac{n^\alpha}{m})^{\beta/(\alpha-1)} + \log \frac{m}{n})$ query time.*

For the d-dimensional halfspace range-counting problem, for example, we have $\alpha = d$ and $\beta = 1 - 1/d$. Thus, for any $n \leq m \leq n^d$, a halfspace range-counting query can be answered in time $O(n/m^{1/d} + \log(m/n))$ using $O(m)$ space.

We conclude this discussion by making a few remarks on Theorem 3.4.

(i) Theorem 3.4 can be refined to balance polylogarithmic factors in the sizes and query times of $\mathcal{D}^1$ and $\mathcal{D}^2$. For example, if the size of $\mathcal{D}^1$ is $O(n^\alpha \operatorname{polylog} n)$ and rest of the parameters are the same as in the theorem, then the query time of the new data structure is $O((\frac{n^\alpha}{m})^{\beta/(\alpha-1)} \operatorname{polylog}(\frac{m}{n}))$. For example, Matoušek [72] showed that for any $n \leq m \leq n^d$, a simplex range-counting query can be answered in time $O((n/m^{1/d}) \log^{d+1}(m/n))$ using $O(m)$ space.

(ii) it is not essential for $\mathcal{D}_1$ or $\mathcal{D}_2$ to be tree-based data structures. It is sufficient to have an efficient, r-convergent decomposition scheme with a partial order on the canonical subsets.

3.4 Lower bounds

Fredman [55] showed that a sequence of n insertions, deletions, and halfplane queries on a set of points in the plane requires $\Omega(n^{4/3})$ time, in the semigroup model. His technique, however, does not

extend to static data structures. In a series of papers, Chazelle has proved nontrivial lower bounds on the complexity of online simplex range searching, using various elegant mathematical techniques. He showed that any data structure of size m, for $n \leq m \leq n^d$, 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 $d \geq 3$ in the worst case. It should be pointed out that this theorem holds even if the query ranges are wedges or strips, but not if the ranges are halfspaces. For halfspaces, Brönnimann *et al.* [23] proved a lower bound of $\Omega((\frac{n}{\log n})^{1-\frac{d-1}{d(d+1)}}/m^{1/d})$ on the query time of any data structure that uses $O(m)$ space, under the semigroup model. Arya *et al.* [18] improved the lower bound to $\Omega((\frac{n}{\log n})^{1-\frac{1}{d+1}}/m^{\frac{1}{d+1}})$. 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 [30] proved an $\Omega(n \log n)$ lower bound for off-line halfspace range searching (i.e., the time taken to answer n queries over a set of n points) under the group model. Exploiting a close-connection between range searching and discrepancy theory, Larsen [64] showed that for any dynamic data structure with t_u and t_q worst-case update and query time, respectively, $t_u \cdot t_q = \Omega(n^{1-1/d})$.

Chazelle and Rosenberg [36] proved a lower bound of $\Omega(\frac{n^{1-\varepsilon}}{m} + k)$ for simplex range reporting under the pointer-machine model. Afshani [1] improved their bound slightly by proving that the size of any data structure that answers a simplex range reporting query in time $O(t_q + k)$ is $\Omega((\frac{n}{t_q})^d/2^{O(\sqrt{\log t_q})})$.

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 $\Omega(n \log m + n^{2/3}m^{2/3} + m \log n)$ [52], 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 [72]. 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 [53]. For d-dimensional hyperplane queries, $\Omega(n^d/\operatorname{polylog} n)$ preprocessing time is required to achieve polylogarithmic query time, and the best possible query time is $\Omega(n^{1/d}/\operatorname{polylog} n)$ if $O(n \operatorname{polylog} n)$ preprocessing time is allowed. For $d = 2$, if the preprocessing time is t_p, the query time is $\Omega(n/\sqrt{t_p})$.

4 Halfspace Range Reporting

A halfspace range-reporting query can be answered more quickly than a simplex range-reporting query using the so-called shallow cuttings and the filtering search technique. We begin by considering a simpler problem: the *halfspace-emptiness* query, which asks whether a query halfspace contains any input point. For simplicity, assume the query halfspace to lie above its bounding hyperplane. 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 $\mathcal{P}(H)$, defined by the intersection of halfspaces lying below the hyperplanes of H. For $d \leq 3$, a point-location query in $\mathcal{P}(H)$ can be answered optimally in $O(\log n)$ time using $O(n)$ space and $O(n \log n)$ preprocessing since $\mathcal{P}(H)$

has linear size [45]. For $d \geq 4$, point-location query in $\mathcal{P}(H)$ becomes more challenging. Clarkson [40] had described a random-sampling based data structure of size $O(n^{\lfloor d/2 \rfloor + \varepsilon})$ that could anser a point-location query in $O(\log n)$ time. Here we describe an approach based on shallow-cutting, which can be viewed as a refinement of Clarkson's approach. Given a parameter $r \in [1, n]$ and an integer $q \in [0, n-1]$, a *shallow* $(q, 1/r)$*-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.

The following theorem by Matoušek can be used to construct a point-location data structure for $\mathcal{P}(H)$:

Theorem 4.1 (Matoušek [70])**.** *Let H be a set of n hyperplanes and $r \leq n$ a parameter. A shallow (0,1/r)-cutting of H of size $O(r^{\lfloor d/2 \rfloor})$ can be computed in time $O(nr^c)$, where c is a constant.*

Choose r to be a sufficiently large constant and compute a shallow $(0,1/r)$-cutting Ξ of H using Theorem 4.1. 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 $O(n^{\lfloor d/2 \rfloor + \varepsilon})$, where $\varepsilon > 0$ is an arbitrarily small constant, and it can be constructed in $O(n^{\lfloor d/2 \rfloor + \varepsilon})$ time. If a query point q does not lie in a simplex of Ξ, then one can conclude that $q \notin \mathcal{P}(H)$ 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 [74] showed that the space can be reduced to $O(\frac{n^{\lfloor d/2 \rfloor}}{\log^{\lfloor d/2 \rfloor - \varepsilon} n})$ while keeping the query time to $O(\log n)$.

A linear-size data structure can be constructed for answering halfspace-emptiness queries by constructing a simplicial partition analogous to Theorem 3.2, as follows. A hyperplane h is called λ*-shallow* if one of the halfspaces bounded by h contains at most λ points of S.

Theorem 4.2 (Matoušek [70])**.** *Let S be a set of n points in $\mathbb{R}^d$ and let $1 \leq r < n$ be a given parameter. Then there exists a family of pairs $\Pi = \{(S_1, \Delta_1), \ldots, (S_m, \Delta_m)\}$ such that each $S_i \subseteq S$ lies inside the simplex Δ_i, $n/r \leq |S_i| \leq 2n/r$, $S_i \cap S_j = \emptyset$ for all $i \neq j$, and the number of simplices of Π crossed by any (n/r)-shallow hyperplane is $O(\log r)$ for $d \leq 3$ and $O(r^{1-1/\lfloor d/2 \rfloor})$ for $d > 3$. If $r \leq n^\alpha$ for some suitable constant $0 < \alpha < 1$, then Π can be constructed in $O(n \log r)$ time.*

Using this theorem, a partition tree for S can be constructed in the same way as for simplex range searching, by choosing $r = n^\alpha$ for some $\alpha < 1/d$. While answering a query for a halfspace h^+, if h^+ crosses more than $O(r^{1-1/\lfloor d/2 \rfloor})$ simplices of the partition Π_v associated with a node v, then we conclude that h^+ is not (n/r)-shallow, i.e., $h^+ \cap S \neq \emptyset$, and thus we stop. Similarly if for a pair (Δ_i, S_i), $\Delta_i \subseteq h^+$, we conclude that $h^+ \cap S \neq \emptyset$ and we stop. Otherwise, the procedure recursively visits the children of v corresponding to the pairs (Δ_i, S_i) for which h crosses Δ_i. The size of the data structure is $O(n)$, the query time is $O(n^{1-1/\lfloor d/2 \rfloor} \operatorname{polylog} n)$, and the preprocessing time is $O(n \log n)$. The query time can be improved to $n^{1-1/\lfloor d/2 \rfloor} 2^{O(\log^* n)}$ by increasing the preprocessing time to $O(n^{1+\varepsilon})$. For even dimensions, Chan's approach [25] can be extended to construct a linear-size partition tree with query time $O(n^{1-1/\lfloor d/2 \rfloor})$.

The halfspace-emptiness data structures can be adapted to answer halfspace range-reporting queries, using the so-called *filtering search* technique introduced by Chazelle [26]. All the data structures mentioned above answer a range-reporting query in two stages. The first stage "identifies" the k points of a query output, in time $f(n)$ that is independent of the output size, and the second stage explicitly reports these points in $O(k)$ time. Chazelle observes that since $\Omega(k)$ time will be

spent in reporting k points, the first stage can compute in $f(n)$ time a superset of the query output of size $O(k)$, and the second stage can "filter" the actual k points that lie in the query range. This observation not only simplifies the data structure but also gives better bounds in many cases, including halfspace range reporting. See [2, 12, 26, 70] for some applications of filtering search.

An optimal halfspace reporting data structure in the plane was proposed by Chazelle *et al.* [34]. They compute *convex layers* $L_1, \ldots, L_m$ of S, where L_i is the set of points lying on the boundary of the convex hull of $S \setminus \bigcup_{j<i} L_j$, and store them in a linear-size data structure, so that a query can be answered in $O(\log n + k)$ time. For $d = 3$, after a series of papers with successive improvements, a linear-size data structure with $O(\log n + k)$ query time was proposed by Afshani and Chan [2] who combine the shallow-cutting with Theorem 4.2 cleverly; this structure can be constructed in $O(n \log n)$ expected time [25].

For $d > 3$, the linear-size data partition tree for halfspace-emptiness queries can be adapted for answering reporting queries as follows. For each node v of the partition tree, we also preprocess the corresponding canonical subset S_v for simplex range searching and store the resulting partition tree as a secondary data structure of v. Because of the simplex range searching data structure being stored at each node of the tree, the total size of the data structure is $O(n \log \log n)$. For a query halfspace h^+, if its boundary hyperplane h crosses more than $O(r^{1-1/\lfloor d/2 \rfloor})$ simplices of Π_v at a node v, then h is not (n_v/r)-shallow, and the simplex range-reporting data structure stored at v is used to report all k_v points of $S_v \cap h^+$ in time $O(n_v^{1-1/d} + k_v) = O(k_v)$; the last equality follows because $r \leq n_v^{1/d}$ and $k_v = \Omega(n_v/r) = \Omega(n_v^{1-1/d})$. Otherwise, for each pair $(S_i, \Delta_i) \in \Pi_v$, if $\Delta_i \subseteq h^+$, it reports all points S_i; and if Δ_i is crossed by h, it recursively visits the corresponding child of v. The overall query time is $O(n^{1-1/\lfloor d/2 \rfloor} + k)$. The size of the data structure was improved to $O(n)$ by Afshani and Chan [2], and the query time for even values of d was improved to $O(n^{1-1/\lfloor d/2 \rfloor} + k)$ [25].

The technique by Afshani for simplex range-reporting lower bound [1] also shows that the size of any halfspace range-reporting data structure in dimension $d(d+3)/2$ has size $\Omega((\frac{n}{t_q})^d / 2^{O(\sqrt{\log t_q})})$.

Finally, we comment that halfspace-emptiness data structures have been adapted to answer halfspace range-counting queries approximately [2, 13, 14, 57, 61, 79]. 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 $|\gamma \cap S| \leq t \leq (1+\delta)|\gamma \cap S|$, where $\delta > 0$ is a constant [2, 3]. For $d > 3$, such a query can be answered in $O((\frac{n}{t})^{1-1/\lfloor d/2 \rfloor} \operatorname{polylog}(n))$ time using linear space [79].

5 Semialgebraic Range Searching

So far we assumed that the ranges were bounded by hyperplanes, but many applications involve ranges bounded by nonlinear functions. For example, a query of the form "for a given point p and a real number r, find all points of S lying within distance r from p" is a range-searching problem in which ranges are balls. As shown below, range searching with balls in $\mathbb{R}^d$ can be formulated as an instance of halfspace range searching in $\mathbb{R}^{d+1}$. So a ball range-reporting (resp. range-counting) query in $\mathbb{R}^d$ can be answered in time $O((n/m^{1/\lceil d/2 \rceil}) \operatorname{polylog} n + k)$ (resp. $O((n/m^{1/(d+1)}) \log(m/n))$), using $O(m)$ space. (Somewhat better performance can be obtained using a more direct approach, which we will describe shortly.) However, data structures for more general ranges seem more challenging.

A natural class of more general ranges can be defined as follows. 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). Specifically, 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. If d, Δ, s are all constants, we refer to the sets in $\Gamma_{d,\Delta,s}$ as *constant-complexity semialgebraic sets*; such sets are sometimes also called *Tarski cells*. 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 assume that the ranges are of the form

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

where f is a $(d + b)$-variate polynomial specifying the type of range (disks, cylinders, cones, etc.), and a is a b-tuple specifying a specific range of the given type (*e.g.*, a specific disk). Let $\Gamma_f = \{\gamma_f(a) \mid a \in \mathbb{R}^b\}$. We will refer to the range-searching problem in which the ranges are from the set Γ_f as the *Γ_f-range searching*. We describe two approaches for Γ_f-range searching.

Linearization. One approach to answer Γ_f-range queries is to use *linearization*, originally proposed by Yao and Yao [85]. We represent the polynomial $f(a, x)$ in the form

$$f(a, x) = \psi_0(a)\varphi_0(x) + \psi_1(a)\varphi_1(x) + \cdots + \psi_\ell(a)\varphi_\ell(x)$$

where $\varphi_0, \ldots, \varphi_\ell, \psi_0, \ldots, \psi_\ell$ are polynomials. A point $x \in \mathbb{R}^d$ is mapped to the point

$$\varphi(x) = [\varphi_0(x), \varphi_1(x), \varphi_2(x), \ldots, \varphi_\ell(x)] \in \mathbb{R}^\ell,$$

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

$$\psi^\#(a) : \{y \in \mathbb{R}^\ell \mid \psi_0(a)y_0 + \psi_1(a)y_1 + \cdots + \psi_\ell(a)y_\ell \geq 0\},$$

where, again, $[y_0, y_1, \ldots, y_\ell]$ are homogeneous coordinates.

The constant ℓ is called the *dimension* of the linearization. Agarwal and Matoušek [7] have described an algorithm for computing a linearization of the smallest dimension under certain assumptions on φ_i's and ψ_i's.

A Γ_f-range query can now be answered using an ℓ-dimensional halfspace range-searching data structure. Thus, for counting queries, we immediately obtain a linear-size data structure with query time $O(n^{1-1/\ell})$ [72], or a data structure of size $O(n^\ell / \log^{\ell-1} n)$ with logarithmic query time [29]. For $d < \ell$, the performance of the linear-size data structures can be improved by exploiting the fact that the points $\varphi(x)$ have only d degrees of freedom. As mentioned at the end of Section 3.2, the query time in this case can be reduced to $O(n^{1-1/\lfloor (d+\ell)/2 \rfloor + \varepsilon})$. It is an open question whether one can similarly exploit the fact that the halfspaces $\psi^\#(a)$ have only b degrees of freedom to reduce the size of data structures with logarithmic query time when $b < \ell$.

Algebraic methods. In cases where the linearization dimension is very large, semialgebraic queries can also be answered using the following more direct approach proposed by Agarwal and Matoušek [7]. Let S be a set of n points in $\mathbb{R}^d$. For each point p_i, we can define a b-variate polynomial $g_i(a) \equiv f(p_i, a)$. Then $\Gamma_f(a) \cap S$ is the set of points p_i for which $g_i(a) \geq 0$. Hence, the problem reduces to point location in the arrangement of algebraic surfaces $g_i = 0$ in $\mathbb{R}^b$. Let G be the set of resulting surfaces. Using a result by Koltun [63], a point-location query in an arrangement of n algebraic surfaces in $\mathbb{R}^b$ can be answered in $O(\log n)$ time using $O(n^{2b-4+\varepsilon})$ space.

Agarwal and Matoušek [7] extended Theorem 3.2 to Tarski cells and showed how to construct partition trees using this extension, obtaining a linear-size data structure with query time $O(n^{1-1/\gamma+\varepsilon})$, where $\gamma = 2$ if $d = 2$ and $\gamma = 2d - 3$ if $d \geq 3$.[1] Extending Matoušek's shallow-cutting based data structure for halfspace range-reporting queries, Sharir and Shaul [81] showed that the query time for Γ_f range-reporting query can be improved in some special cases.

A better linear-size data structure has been proposed [8, 73] based on the *polynomial partitioning scheme* introduced by Guth and Katz [56]. 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). Guth and Katz show that an r-partitioning polynomial of degree $O(r^{1/d})$ always exists. Agarwal *et al.* [8] described a randomized algorithm to compute such a polynomial in expected time $O(nr + r^3)$. Recent results in real algebraic geometry [20] imply 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 roughly $n^{1-1/d}$. However, this ignores the points in $S^* = S \cap Z(f)$. Use a scheme based on the so-called cylindrical algebraic decomposition, Agarwal *et al.* [8] project the points of S^* to $\mathbb{R}^{d-1}$ and recursively construct a $(d-1)$-dimensional data structure to preprocess S^*. A more elegant and simpler method was subsequently proposed by Matoušek and Patáková [73], which basically applies a generalized polynomial-partitioning scheme on S^* and $Z(f)$. Roughly speaking, they choose another parameter r' and partition S^* further by another polynomial g of degree $O(r'^{1/d})$ such that $Z(f, g) = Z(f) \cap Z(g)$ has dimension $d - 2$ and each component of $Z(f) \setminus Z(g)$ contains at most n^*/r' points of S^*. If $Z(f, g)$ also contains many points, then they partition $Z(f, g)$ by another polynomial h so that $Z(f, g, h)$ has dimension $d - 3$, and so on. The main contribution of their paper is proving the existence of such functions g and h and an algorithm for computing them. Putting everything together, a semialgebraic range-counting query can be answered in $O(n^{1-1/d} \operatorname{polylog}(n))$ 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.

We conclude this section by noting that Arya and Mount [17] have presented a linear-size data structure for approximate range-searching queries. Let γ be a constant-complexity semialgebraic set and $\varepsilon > 0$ a parameter. Their data structure returns in $O(\frac{1}{\varepsilon^d} \log n + k_\varepsilon)$ time a subset S_ε of k_ε points such that $\gamma \cap S \subseteq S_\varepsilon \subseteq \gamma_\varepsilon \cap S$ where γ_ε is the set of points within distance $\varepsilon \cdot \operatorname{diam}(\gamma)$ of γ. If γ is convex, the query time improves to $O(\log n + \frac{1}{\varepsilon^{d-1}} + k_\varepsilon)$. A result by Larsen and Nguyen [65] implies that query time of a linear-size data structure is $\Omega(\log n + \varepsilon^{-\frac{d}{1+\delta}-1})$ for any arbitrarily small

[1] The paper by Agarwal and Matoušek [7] predates the result by Koltun [63]. Using Koltun's result, the value of γ can be improved to $\gamma = d$ for $d \leq 3$ and $\gamma = 2d - 4$ for $d > 3$.

constant $\delta > 0$. The data structure in [17] can also return a value k_ε, with $|S \cap \gamma| \leq k_\varepsilon \leq |S \cap \gamma_\varepsilon|$ in time $O(\frac{1}{\varepsilon^d} \log n)$, or in $O(\log n + \frac{1}{\varepsilon^{d-1}})$ time if γ is convex. See also [44]. Chazelle *et al.* [35] studied the approximate halfspace range-counting problem in high dimensions, where d is not a constant, under a similar notion of approximation — the points within distance ε from the boundary hyperplane may be misclassified. They presented a data structure of size $dn^{O(\varepsilon^{-2})}$ that can answer a query in time $O((d/\varepsilon^2)\,\mathrm{polylog}(d/\varepsilon))$.

6 Variants and Extensions

In this section we review some extensions of range-searching data structures, including multi-level data structures, semialgebraic range searching, and dynamization. As in the previous section, the preprocessing time for each of the data structures we describe is at most a polylogarithmic or n^ε factor larger than its size.

6.1 Multi-level data structures

A rather powerful property of data structures based on decomposition schemes (described in Section 2) is that they can be cascaded together to answer more complex queries, at the increase of a logarithmic factor in their performance. This property has been implicitly used for a long time; see, for example, [49, 66, 80]. The real power of the cascading property was first observed by Dobkin and Edelsbrunner [46], 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; see [9, 25, 72]. In this subsection we briefly sketch the general cascading scheme, as described in [72].

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

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

The $O(\mathrm{polylog}(n))$ query-time data structures for simplex range counting fit in this framework.

For example, a data structure for counting the number of points in a wedge (i.e., intersection of two halfspaces) in $\mathbb{R}^d$ can be constructed by cascading two d-dimensional halfspace range-counting data structures, as follows. Let S be a set of n points. We define two binary relations $\Diamond^1$ and $\Diamond^2$, where for any wedge $\gamma = \gamma_1 \cap \gamma_2$, where γ_1, γ_2 are half-spaces, $p \Diamond^i \gamma$ if $p \in \gamma_i$ $(i = 1, 2)$. Let $\mathcal{P}^i$ be the searching problem associated with $\Diamond^i$, and let $\mathcal{D}^i$ be the halfspace range-counting data structure corresponding to $\mathcal{P}^i$. Then the wedge range-counting problem is the same as $\mathcal{P}^1 \circ \mathcal{P}^2$. We can therefore cascade $\mathcal{D}^1$ and $\mathcal{D}^2$, as described above, to answer a wedge range-counting query. Similarly, a data structure for d-dimensional simplex range-counting can be constructed by cascading $d + 1$ halfspace range-counting data structures.

The following theorem, whose straightforward proof we omit, states a general result for multi-level data structures.

Theorem 6.1. *Let $S, \mathcal{P}^1, \mathcal{P}^2, \mathcal{D}^1, \mathcal{D}^2$ be as defined above, and let r be a constant. Suppose the size and query time of each decomposition scheme are at most $S(n)$ and $Q(n)$, respectively. If $\mathcal{D}^1$ is efficient and r-convergent, then we obtain a hierarchical decomposition scheme $\mathcal{D}$ for $\mathcal{P}^1 \circ \mathcal{P}^2$ whose size and query time are $O(S(n) \log_r n)$ and $O(Q(n) \log_r n)$. If $\mathcal{D}^2$ is also efficient and r-convergent, then $\mathcal{D}$ is also efficient and r-convergent.*

For example, the wedge range-counting data structure described above has $O(\frac{n^d}{\log^{d-2} n})$ space and $O(\log^2 n)$ query time, and a simplex range counting query can be answered in $O(\log^{d+1} n)$ time using $O(n^d)$ space [72].

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. The next two subsections will describe a few multi-level data structures.

6.2 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 $\sum_{p \cap \gamma \neq \emptyset} w(p)$, where the sum is taken over all objects $p \in S$ that intersect γ, can be computed quickly. Range searching is a special case of intersection searching in which S is a set of points.

Efficient data structures for many intersection-searching problems have been developed by expressing the intersection test as a conjunction of simple primitive tests (in low dimensions) and using a multi-level data structure to perform these tests. For example, a segment γ intersects another segment e if the endpoints of e lie on the opposite sides of the line containing γ and vice-versa. Suppose we want to report those segments of S whose left endpoints lie below the line supporting a query segment (the other case can be handled in a similar manner). We define three searching problems $\mathcal{P}^1, \mathcal{P}^2$, and $\mathcal{P}^3$, with relations $\Diamond^1, \Diamond^2, \Diamond^3$, as follows:

$e \Diamond^1 \gamma$**:** The left endpoint of e lies below the line supporting γ.

$e \Diamond^2 \gamma$**:** The right endpoint of e lies above the line supporting γ.

$e \diamondsuit^3 \gamma$**:** The line ℓ_e supporting e intersects γ; equivalently, in the dual plane, the point dual to ℓ_e lies in the double wedge dual to e.

For $1 \leq i \leq 3$, let $\mathcal{D}^i$ denote a data structure for $\mathcal{P}^i$. Then $\mathcal{D}^1$ (resp. $\mathcal{D}^2$) is a halfplane range-searching structure on the left (resp. right) endpoints of segments in S, and $\mathcal{D}^3$ is (essentially) a triangle range-searching structure for points dual to the lines supporting S. By cascading $\mathcal{D}^1$, $\mathcal{D}^2$, and $\mathcal{D}^3$, we obtain a data structure for segment-intersection queries. Therefore, by Theorem 6.1, a segment-intersection query can be answered in time $O(n^{1/2}\log^2 n)$ using $O(n\log^2 n)$ space, or in $O(\log^2 n)$ time using $O(n^2 \log n)$ space. Similarly, the condition for a query point p lying inside a triangle can be expressed as the conjunction of three tests, each testing whether p lies in a halfplane. So point-stabbing queries amid triangles, i.e., counting all input triangles that intersect a query point, can also be answered in $O(\sqrt{n}\log^2 n)$ (resp. $O(\log^2 n)$) time using $O(n\log^2 n)$ (resp. $O(n^2 \log n)$) space. Table 1 summarizes a few intersection searching results.

d	Objects	Range	Size	Query Time	Source
$d=2$	Disk	Point	m	$(n/\sqrt{m})^{4/3}$	[25]
	Segment	Segment	m	$n/\sqrt{m}$	[72, 25]
	Triangle	Point	m	$n/\sqrt{m}$	[72, 25]
	Circular arc	Segment	m	$n/m^{1/3}$	[12]
$d=3$	Tetrahedron	Point	m	$n/m^{1/3}$	[72]
	Sphere	Segment	m	$n/m^{1/3}$	[7]
	Triangle	Segment	m	$n/m^{1/4}$	[7]
$d>3$	Simplex	Point	m	$n/m^{1/d}$	[72]

Table 1. Asymptotic upper bounds for intersection-counting queries, with polylogarithmic factors omitted. Reporting queries can be answered by paying an additional $O(k)$ cost.

6.3 Ray-shooting queries

Preprocess a set S of objects in $\mathbb{R}^d$ into a data structure so that the first object (if any) hit 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 [77], was proposed by Agarwal and Matoušek [6]. Suppose we have a segment intersection-detection data structure for S. Let ρ be a query ray. Their algorithm maintains a segment $\overline{ab} \subseteq \rho$ such that the first intersection point of $\overline{ab}$ with S is the same as that of ρ. If a lies on an object of S, it returns a. Otherwise, it picks a point $c \in \overline{ab}$ and determines, using the segment intersection-detection data structure, whether the interior of the segment $\overline{ac}$ intersects any object of S. If the answer is yes, it recursively finds the first intersection point of $\overline{ac}$ with S; otherwise, it recursively finds the first intersection point of $\overline{cb}$ with S. Using parametric searching, the point c at each stage can be chosen in such a way that the algorithm terminates after $O(\log n)$ steps. In some cases, the query time can be improved by a polylogarithmic factor by exploiting the additional structure of input objects, e.g., bypassing the need for parametric search.

Another approach for answering ray-shooting queries is the locus approach. A ray in $\mathbb{R}^d$ can be represented as a point in $\mathbb{R}^d \times \mathbb{S}^{d-1}$. Given a set S of objects, we can partition the parametric

space $\mathbb{R}^d \times \mathbb{S}^{d-1}$ into cells so that all points within each cell correspond to rays that hit the same object first; this partition is called the *visibility map* of S. Using this approach and some other techniques, Chazelle and Guibas [33] showed that a ray-shooting query in a simple polygon can be answered in $O(\log n)$ time using $O(n)$ space. A simpler data structure was subsequently proposed by Hershberger and Suri [59].

d	Objects	Size	Query Time	Source
$d = 2$	Simple polygon	n	$\log n$	[59]
	s disjoint simple polygons	n	$\sqrt{s}$	[10, 59]
	s disjoint convex polygons	$s^2 + n$	$\log n$	[78]
	Segments	m	$n/\sqrt{m}$	[9, 39]
	Circular arcs	m	$n/m^{1/3}$	[12]
	Disjoint arcs	n	$\sqrt{n}$	[12]
$d = 3$	Convex polytope	n	$\log n$	[47]
	s convex polytopes	$s^2 n^{2+\varepsilon}$	$\log^2 n$	[11, 62]
	Halfplanes	m	$n/m^{1/3}$	[6]
	Triangles	m	$n/m^{1/4}$	[7]
	Spheres	m	$n/m^{1/3}$	[7, 81]
$d > 3$	Hyperplanes	m	$n/m^{1/d}$	[6]
	Convex polytope	m	$n/m^{1/\lfloor d/2 \rfloor}$	[6, 75]

Table 2. Asymptotic upper bounds for ray shooting queries, with polylogarithmic factors omitted.

Table 2 gives a summary of known ray-shooting results. For the sake of clarity, we have omitted polylogarithmic factors from query times of the form n/m^{α}.

6.4 Nearest-neighbor queries

The *nearest-neighbor (NN) query* problem is defined as follows: Preprocess a set S of points in $\mathbb{R}^d$ into a data structure so that a point in S closest to a query point ξ can be reported quickly. This is one of the most widely studied problems not only in computational geometry but in many other fields such as machine learning, computer vision, database systems, information retrieval, and geographic information systems.

For simplicity, we assume that the distance between points is measured in the Euclidean metric, though a more complicated metric can be used depending on the application. For $d = 2$, one can construct the Voronoi diagram of S and answer NN queries by performing point-location queries in the Voronoi diagram. This approach gives an optimal data structure with $O(\log n)$ query time, $O(n)$ size, and $O(n \log n)$ preprocessing [45]. No such data structure is known for even $d = 3$. A NN query for a set of points under the Euclidean metric in $\mathbb{R}^d$ can be formulated as an instance of the ray-shooting problem in a convex polyhedron in $\mathbb{R}^{d+1}$, as follows. We map each point $p = (p_1, \ldots, p_d) \in S$ to a hyperplane $\hat{p}$ in $\mathbb{R}^{d+1}$, which is the graph of the function

$$f_p(x_1, \ldots, x_d) = 2p_1x_1 + \cdots + 2p_dx_d - (p_1^2 + \cdots + p_d^2).$$

Then p is a closest neighbor of a point $\xi = (\xi_1, \ldots, \xi_d)$ if and only if

$$f_p(\xi_1, \ldots, \xi_d) = \max_{q \in S} f_q(\xi_1, \ldots, \xi_d).$$

That is, if and only if f_p is the first hyperplane intersected by the vertical ray $\rho(\xi)$ emanating from the point $(\xi_1, \ldots, \xi_d, 0)$ in the negative x_{d+1}-direction. If we define $P = \bigcap_{p \in S} \{(x_1, \ldots, x_{d+1}) \mid$

$x_{d+1} \geq f_p(x_1, \ldots, x_d)\}$, then p is the nearest neighbor of ξ if and only if the intersection point of $\rho(\xi)$ and ∂P lies on the graph of f_p. Thus a nearest-neighbor query can be answered in time roughly $n/m^{1/\lceil d/2 \rceil}$ using $O(m)$ space [41, 6, 75]. This approach can be extended to answer farthest-neighbor and k-nearest-neighbor queries also. In general, if we have an efficient data structure for answering disk-emptiness queries for disks under a given metric ρ, we can apply parametric searching to answer nearest-neighbor queries under the ρ-metric, provided the data structure satisfies certain mild assumptions [6].

Since answering NN queries is expensive even for moderate values of d, there is extensive work on computing ε-approximate nearest neighbors, i.e., returning a point $\tilde{p} \in S$ such that $\|\xi\tilde{p}\| \leq (1+\varepsilon)\min_{p \in S} \|\xi p\|$, for a given parameter $\varepsilon > 0$; see e.g. [16, 15] and the references therein. For a parameter $\theta \in [2, 1/\varepsilon]$, an ε-approximate NN query can be answered in $O(\log(n\theta) + \frac{1}{(\varepsilon\theta)^{(d-1)/2}})$ time using $O(n\theta^{d-1}\log\frac{1}{\varepsilon})$ space. For $\theta = 2$, this yields a data structure of size $O(n\log\frac{1}{\varepsilon})$ with $O(\log n + \frac{1}{\varepsilon^{(d-1)/2}})$ query time, and for $\theta = 1/\varepsilon$, it yields a data structure of size $O(\frac{n}{\varepsilon^{d-1}}\log\frac{1}{\varepsilon})$ with $O(\log\frac{n}{\varepsilon})$ query time. The performance of these data structures depends exponentially on d, so they are not efficient for large values of d. There is much work on data structures for approximate NN-queries whose query time and size have polynomial dependence on d; see [60] for a survey of higher dimensional NN queries.

6.5 Linear-programming queries

Let S be a set of n halfspaces in $\mathbb{R}^d$. We wish to preprocess S into a data structure so that for a direction vector $\vec{v}$, we can determine the first point of $\bigcap_{h \in S} h$ in the direction $\vec{v}$. For $d \leq 3$, such a query can be answered in $O(\log n)$ time using $O(n)$ storage, by constructing the normal diagram of the convex polytope $\bigcap_{h \in S} h$ and preprocessing it for point-location queries. For higher dimensions, Matoušek [71] showed that, using multidimensional parametric searching and a data structure for answering halfspace emptiness queries, a linear-programming query can be answered in $O((n/m^{1/\lfloor d/2 \rfloor})\operatorname{polylog} n)$ with $O(m)$ storage. Chan [24] described a randomized procedure whose expected query time is $n^{1-1/\lfloor d/2 \rfloor}2^{O(\log^* n)}$, using linear space.

7 Concluding Remarks

In this chapter we reviewed data structures for range searching and a few related problems. The quest for efficient range-searching data structures has resulted in several elegant geometric techniques that have enriched computational geometry as a whole. It would have been impossible to describe all techniques and results on range searching and their applications, so we focused on a few of them. The emphasis of the chapter was on the known results, and we did not make an attempt to list open problems. We conclude this chapter by mentioning a few open problems:

- Although near-optimal bounds are known for simplex range searching in the semigroup model, the known lower bounds in the group model are far from optimal. Also, can a lower bound of $n/m^{1/2}$ be proved for simplex range counting in the real RAM model of computation?
- Can a semialgebraic range query in $\mathbb{R}^d$ be answered in $O(\log n)$ time using $O(n^d)$ space? The simplest question in this direction is whether a disk range-counting query in $\mathbb{R}^2$ be answered in $O(\log n)$ time using $O(n^2)$ space?

- From a practical standpoint, simplex range searching is still largely open; known data structures are rather complicated and do not perform well in practice. Lower bounds suggest that we cannot hope for data structures that do significantly better than the naïve algorithm in the worst case (and for some problems, even in the average case). An interesting open question is to develop simple data structures that work well under some assumptions on input points and query ranges.

References

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

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

[3] 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 (2010), 700–712.

[4] P. K. Agarwal, Range searching, in: *Handbook of Discrete and Computational Geometry* (J. E. Goodman, J. O'Rourke, and C. Toth, eds.), CRC Press LLC, Boca Raton, FL, 2016, p. to appear.

[5] P. K. Agarwal and J. Erickson, Geometric range searching and its relatives, in: *Advances in Discrete and Computational Geometry* (B. Chazelle, J. E. Goodman, and R. Pollack, eds.), *Contemporary Mathematics*, Vol. 223, American Mathematical Society, Providence, RI, 1999, pp. 1–56.

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

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

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

[9] P. K. Agarwal and M. Sharir, Applications of a new space-partitioning technique, *Discrete Comput. Geom.*, 9 (1993), 11–38.

[10] P. K. Agarwal and M. Sharir, Ray shooting amidst convex polygons in 2D, *J. Algorithms*, 21 (1996), 508–519.

[11] P. K. Agarwal and M. Sharir, Ray shooting amidst convex polyhedra and polyhedral terrains in three dimensions, *SIAM J. Comput.*, 25 (1996), 100–116.

[12] P. K. Agarwal, M. van Kreveld, and M. Overmars, Intersection queries in curved objects, *J. Algorithms*, 15 (1993), 229–266.

[13] B. Aronov and S. Har-Peled, On approximating the depth and related problems, *SIAM J. Comput.*, 38 (2008), 899–921.

[14] B. Aronov and M. Sharir, Approximate halfspace range counting, *SIAM J. Comput.*, 39 (2010), 2704–2725.

[15] S. Arya, T. Malamatos, and D. M. Mount, Space-time tradeoffs for approximate nearest neighbor searching, *J. ACM*, 57 (2009).

[16] S. Arya and D. M. Mount, Approximate nearest neighbor queries in fixed dimensions, *Proc. 4th ACM-SIAM Sympos. Discrete Algorithms*, 1993, pp. 271–280.

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

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

[19] D. Avis, Non-partitionable point sets, *Inform. Process. Lett.*, 19 (1984), 125–129.

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

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

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

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

[24] T. M. Chan, Fixed-dimensional linear programming queries made easy, *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, 1996, pp. 284–290.

[25] T. M. Chan, Optimal partition trees, *Discrete Comput. Geom.*, 47 (2012), 661–690.

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

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

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

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

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

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

[32] B. Chazelle and J. Friedman, A deterministic view of random sampling and its use in geometry, *Combinatorica*, 10 (1990), 229–249.

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

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

[35] B. Chazelle, D. Liu, and A. Magen, Approximate range searching in higher dimension, *Comput. Geom.: Theory Appls.*, 39 (2008), 24–29.

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

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

[38] B. Chazelle and E. Welzl, Quasi-optimal range searching in spaces of finite VC-dimension, *Discrete Comput. Geom.*, 4 (1989), 467–489.

[39] S. W. Cheng and R. Janardan, Algorithms for ray-shooting and intersection searching, *J. Algorithms*, 13 (1992), 670–692.

[40] K. L. Clarkson, New applications of random sampling in computational geometry, *Discrete Comput. Geom.*, 2 (1987), 195–222.

[41] K. L. Clarkson, A randomized algorithm for closest-point queries, *SIAM J. Comput.*, 17 (1988), 830–847.

[42] R. Cole, Partitioning point sets in 4 dimensions, *Proc. 12th Internat. Colloq. Automata Lang. Program.*, *Lecture Notes Comput. Sci.*, Vol. 194, Springer-Verlag, 1985, pp. 111–119.

[43] R. Cole and C. K. Yap, Geometric retrieval problems, *Inform. Control*, 63 (1984), 39–57.

[44] G. D. da Fonseca and D. M. Mount, Approximate range searching: The absolute model, *Comput. Geom.: Theory Appls.*, 43 (2010), 434–444.

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

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

[47] D. P. Dobkin and D. G. Kirkpatrick, A linear algorithm for determining the separation of convex polyhedra, *J. Algorithms*, 6 (1985), 381–392.

[48] H. Edelsbrunner, D. G. Kirkpatrick, and H. A. Maurer, Polygonal intersection searching, *Inform. Process. Lett.*, 14 (1982), 74–79.

[49] H. Edelsbrunner and H. A. Maurer, A space-optimal solution of general region location, *Theoret. Comput. Sci.*, 16 (1981), 329–336.

[50] H. Edelsbrunner and E. Welzl, Halfplanar range search in linear space and $O(n^{0.695})$ query time, *Inform. Process. Lett.*, 23 (1986), 289–293.

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

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

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

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

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

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

[57] S. Har-Peled and M. Sharir, Relative (p, ϵ)-approximations in geometry, *Discr. Comput. Geom.*, 45 (2011), 462–496.

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

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

[60] P. Indyk, Nearest neighbors in high-dimensional spaces, in: *Handbook of Discrete and Computational Geometry* (J. E. Goodman, J. O'Rourke, and C. Toth, eds.), CRC Press LLC, Boca Raton, FL, 2016, p. to appear.

[61] H. Kaplan, E. Ramos, and M. Sharir, Range minima queries with respect to a random permutation, and approximate range counting, *Discr. Comput. Geom.*, 45 (2011), 3–33.

[62] H. Kaplan, N. Rubin, and M. Sharir, Linear data structures for fast ray-shooting amidst convex polyhedra, *Algorithmica*, 55 (2009), 283–310.

[63] V. Koltun, Sharp bounds for vertical decompositions of linear arrangements in four dimensions, *Discr. Comput. Geom.*, 31 (2004), 435–460.

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

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

[66] G. S. Lueker, A data structure for orthogonal range queries, *Proc. 19th Annu. IEEE Sympos. Found. Comput. Sci.*, 1978, pp. 28–34.

[67] J. Matoušek, Construction of ϵ-nets, *Discrete Comput. Geom.*, 5 (1990), 427–448.

[68] J. Matoušek, Cutting hyperplane arrangements, *Discrete Comput. Geom.*, 6 (1991), 385–406.

[69] J. Matoušek, Efficient partition trees, *Discrete Comput. Geom.*, 8 (1992), 315–334.

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

[71] J. Matoušek, Linear optimization queries, *J. Algorithms*, 14 (1993), 432–448.

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

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

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

[75] J. Matoušek and O. Schwarzkopf, On ray shooting in convex polytopes, *Discrete Comput. Geom.*, 10 (1993), 215–232.

[76] J. Matoušek, *Using the Borsuk-Ulam Theorem*, Springer, Heidelberg, 2003.

[77] N. Megiddo, Applying parallel computation algorithms in the design of serial algorithms, *J. ACM*, 30 (1983), 852–865.

[78] M. Pocchiola and G. Vegter, Pseudo-triangulations: Theory and applications, *Proc. 12th Annu. ACM Sympos. Comput. Geom.*, 1996, pp. 291–300.

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

[80] H. W. Scholten and M. H. Overmars, General methods for adding range restrictions to decomposable searching problems, *J. Symbolic Comput.*, 7 (1989), 1–10.

[81] M. Sharir and H. Shaul, Semialgebraic range reporting and emptiness searching with applications, *SIAM J. Comput.*, 40 (2011), 1045–1074.

[82] E. Welzl, Partition trees for triangle counting and other range searching problems, *Proc. 4th Annu. ACM Sympos. Comput. Geom.*, 1988, pp. 23–33.

[83] D. E. Willard, Polygon retrieval, *SIAM J. Comput.*, 11 (1982), 149–165.

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

[85] A. C. Yao and F. F. Yao, A general approach to D-dimensional geometric queries, *Proc. 17th Annu. ACM Sympos. Theory Comput.*, 1985, pp. 163–168.

[86] F. F. Yao, A 3-space partition and its applications, *Proc. 15th Annu. ACM Sympos. Theory Comput.*, 1983, pp. 258–263.

[87] F. F. Yao, D. P. Dobkin, H. Edelsbrunner, and M. S. Paterson, Partitioning space for range queries, *SIAM J. Comput.*, 18 (1989), 371–384.