Non-orthogonal Range Searching: A Review

Technical Report No. 2008-546

Yurai Núñez-Rodríguez
School of Computing
Queen's University
Kingston, ON, Canada
yurai@cs.queensu.ca

Abstract

Range searching is a common type of database query where the user enquires about the elements that lie within a certain geometric shape. Typically, a query shape is defined by a simple geometric object, such as a rectangle, a disk, a triangle, a parametric curve, or a combination of them. In particular, non-orthogonal range searching covers the most general cases of range searching since the query shape is not constrained to be a rectangle parallel to the axis coordinates. We review the long history of non-orthogonal range searching from its beginning to the present and show the latest advances in this area. Most traditional scenarios of non-orthogonal range searching have been well studied and the available algorithms are near-optimal. Latest results include heuristic algorithms, small improvements for higher dimensions, and new problems on range searching with a variety of applications.

1 Introduction

A common type of database query consists of searching according to a range of key values. Such queries are termed *range queries*, and the problem associated with this type of queries is known as *range searching*. Ranges are often defined in multiple dimensions; for example, a range query could be phrased: "Report all the people in the database who live no more than 1 km away from the market place, are between 40 and 60 years old, and whose names fall between John and Joseph in alphabetical order. This problem has a long history, and has been widely studied by the Computational Geometry community. Range searching has multiple applications on relational databases, Geographic Information Systems (GIS), Computer Graphics, networks and communication, and other areas, including other problems in Computational Geometry itself.

Range queries can be easily carried out by fetching all the features in the database and reporting those matching the search criterion. It is assumed at all times that checking whether an object is in a given range can be done in constant time. This naive approach offers linear time complexity, which may be a prohibited cost in large datasets, especially when many queries are executed in batches. Several data structures and

algorithms have been developed to solve this problem more efficiently; the most relevant will be reviewed during the course of this survey.

1.1 Notation and general definitions

The following are some terms and definitions that we will use throughout this paper:

- the prefix *hyper*: Refers to higher dimensional features which are analogous to their lower dimensional homologues. For instance, hyperplanes are homologues of planes, but in dimensions 4 or higher. Similarly we use hypercube, hyperpyramid, etc.
- *halfplane*: Half of the plane bounded by a straight line
- *halfspace*: Half of the space bounded by a hyperplane
- *polylogarithm*: Function bounded by $O(\log^c n)$ for some constant c. It appears as "polylog n" in mathematical expressions

The general setting for range searching is defined by a pair (P, R) which consists of a set of points $P = \{p_i \in \mathbb{R}^d, i = 1..n\}$, and R a family of subsets of $\mathbb{R}^d$ called *ranges*. Examples of range families are the set of all halfplanes in $\mathbb{R}^2$, and the set of volumes bounded by tetrahedrons in $\mathbb{R}^3$. By $\mathbb{R}^d$, we mean the d-dimensional real space, with distances according to the Euclidean metric. For range searching on other metric spaces see the survey by Chávez *et al.* [38]. In the following, we refer to a pair (P, R) as a *range space*.

We are concerned about the complexity of solving range searching problems. In particular, we are more interested in $Q(n, d)$, the query time, $S(n, d)$, the space or size of the data structure, and $P(n, d)$, the preprocessing time or time it takes to build the corresponding data structure, although the latter is not considered as relevant since it only affects the initial stage. Other important efficiency criteria, such as the cost of updating the data structures and the number of accesses to secondary memory, will be included later in this report.

1.2 Different variants of range searching

We will review *geometric range searching* in this survey. Geometric range searching, referred to as range searching in the following, is a special case of range searching where the space dimension is relatively small, say between 2 and 20, the input points are embedded in a continuous space ($\mathbb{R}^d$ as seen before), and the ranges have a geometric shape. Probably the most well known problem, in that sense, is *orthogonal range searching*. For orthogonal range searching the ranges consist of *d-rectangles* or the cross product of a set of intervals in d dimensions. Other types of ranges include *halfspaces*, *simplices*[1], and *balls*. Section 3.10, will look at more general ranges.

[1] A simplex in $\mathbb{R}^d$ is a volume bounded by $d + 1$ hyperplanes of dimension $d - 1$ (e.g. triangles in $\mathbb{R}^2$, and tetrahedrons in $\mathbb{R}^3$).

A range query may attempt to answer different questions, for example, what is the set of points contained in the query range (*range reporting*), what is the number of points in the range (*range counting*), does the range contain any point (*emptiness query*), or which element has maximum *weight* of those contained in the range (*extremal queries*). For the latter case the weights are given by a function in the input points.

Queries can be performed on-line or off-line. By on-line we mean that the set of queries is not known in advance, not even the number of them. So, a data structure needs to be built on the basis of efficiently answering any number of range queries of a given type. In this case, we assume that there will be an infinite number of queries, so the preprocessing time is not as critical as the query time or the size of the data structure. In the off-line case, all queries are known beforehand and the algorithms are tailored to answer that specific set of queries.

The rest of this survey focuses on low-dimensional non-orthogonal on-line range searching primarily; although we may present results from other types of range searching for comparison. Because of space limitations we do not include approximate range searching or dynamic data structures in this survey.

1.3 About this survey

Excellent surveys have been written in the past by Matoušek [74], and Agarwal and Erickson [2]. However, these surveys do not include newer results or applications of range searching that have originated in the last 8 years. This work is intended to include both older and newer aspects of range searching. We also intend to explain range searching in simpler words and illustrate with figures what may seem complicated in words. We have formally and informally defined basic concepts needed for the understanding of range searching and made this survey a self-contained document.

The rest of this document is structured as follows: section 2 presents lower bounds for range searching on different computational models, section 3 reviews classical range searching algorithms, section 4 offer a view on data structures that may not be comparable in theoretical terms but have proved to perform well in practise, and section 5 is an overview of newer variants of range searching and potential research directions, with special attention to Wireless Sensor Networks. We finalize with some concluding remarks on the topics covered.

2 Lower bounds

When designing data structures and algorithms to solve a given problem, it is crucial to know how efficiently the problem can be solved. This is the purpose of lower bounds: they provide a bound on the minimum number of operations or memory units needed to solve the problem. Usually a lower bound refers to the worst case that one can encounter over all the possible inputs, or the average case under certain assumptions on the expected input. A good lower bound is one that matches, or nearly matches (up to polylogarithmic factor), the asymptotic complexity of the corresponding upper bounds. The upper bounds are provided by known algorithms; in section 3, we will review several optimal or near optimal algorithms. To formally obtain lower bounds, a formal definition of algorithm is required. For this purpose, several models of computation have been developed.

Range	Problem	Model	Q(n,d)	Source
simplex	semigroup	semigroup ($d = 2$)	$\frac{n}{\sqrt{m}}$	[22]
simplex	semigroup	semigroup ($d > 2$)	$\frac{n}{m^{1/d} \log n}$	[22]
simplex	reporting	pointer machine	$\frac{n^{1-\epsilon}}{m^{1/d}+k}$	[33]
halfspace	semigroup	semigroup	$(\frac{n}{\log n})^{\frac{d^2+1}{d^2+d}} m^{-1/d}$	[20]
halfspace	semigroup	integral semigroup	$(\frac{n}{\log n})/m^{\frac{d+1}{d^2+1}}$	[11]
halfspace	emptiness	partition graph	$(\frac{n}{\log n})^{\frac{\delta^2+1}{\delta^2+d}} m^{-1/\delta}$, where $d \geq \delta(\delta+3)/2$	[47]

Table 1: Lower bounds for different versions of the static on-line range searching problem using O(m) storage.

One model of computation on which lower bounds first appeared is the *decision tree* model. This model consists of a k-ary tree, whose nodes contain k children and a question about the input. Each child represents one possible answer to the associated question. The computation in a decision tree starts from the root and proceeds with the child node corresponding to the right answer at each step. When a leaf node is reached, the answer to the problem is the value stored in it. There are several versions of decision trees. In the *standard decision tree*, the question asked at the nodes is a simple comparison, usually with only two or three possible answers, depending on whether equality is considered. For a survey on decision trees and other models of computations, see Erickson's PhD dissertation [46]. However, the decision tree models do not provide a good lower bound for range searching problems. Lueker [67] showed how the (standard) linear decision tree is not powerful enough to obtain tight lower bounds in a dimension higher than two.

Another well known model of computation is the *random access machine* (RAM) model. This model is an abstraction of a real computer. Its memory is defined as an array of cells that can be accessed in constant time. The cells can store integer values of up to $\log n$ bits (n being the size of the input), and arithmetic operations and comparisons can be used. A more powerful version of RAM is the *real* RAM where infinite precision is allowed. The RAM and real RAM models are precisely defined by Tarjan [7] and Aho *et al.* [80] respectively. RAM models are too complicated to derive lower bounds proofs in a relatively easy way. These models are better suited for describing algorithms and upper bounds.

In the following, we will examine the models of computation that have been used to obtain nontrivial lower bounds for range searching. Table 1 summarizes the best known lower bounds for different variations of non-orthogonal range searching problems. Many of the following results for the non-orthogonal case have their orthogonal counterparts [53, 23, 24] as well.

2.1 The arithmetic model

In the arithmetic model of computation, the points are assigned weights from a commutative semigroup $(W, \oplus)$[2] according to a weight function $w : w(p) \in W, \forall p \in P$. The result of a range query q is then expressed as the total weight of the points in the range $(w(P \cap q))$. This is a generalization for many types

[2]A semigroup $(W, \oplus)$ is a set W with an associative addition operator $\oplus : W \times W \rightarrow W$. A semigroup is commutative if $x \oplus y = y \oplus x$ for all $x, y \in W$.

of range queries. For instance, consider the semigroup $(2^P, \cup)$. If each point p is assigned a weight that consist of the point itself ($w(p) = p$), the *cumulative weight* of the points that satisfy a given range query corresponds to the answer of the reporting case ($w(P \cap q) = P \cap q$). Similarly, it is easily seen that for semigroups $(\mathbb{R}, +)$, $(\{0, 1\}, \vee)$, and $(\mathbb{R}, max)$ there are weight functions such that the cumulative weights correspond to the answers of range counting, emptiness queries, and extremal queries respectively.

In this model, given a range query, we only account for the number of sums required to compute the total weight of the answer to a given range query, which is a lower bound for $Q(n, d)$. However, many sums can be precomputed and stored in advance. When a query is performed, the stored values are used to compute the answer in a shorter time. Thus, $S(n, d)$ and $P(n, d)$ will be bounded by the number of precomputed sums stored in memory. The sets for which sums have been computed in advance are usually called *canonical sets* or *clusters* whereas the linear forms or precomputed sums are called *generators*.

Fredman [53] was the first to use the arithmetic model to obtain lower bounds for the general range searching problem. He proved that queries, insertions, and deletions take $\Omega(n^{1/3})$ average time on any data structure that stores semigroup weights [54]. This is applicable to halfspace range searching and other types of ranges. His results are based on the relationship between the set of generators used for answering a query and the set of generators that are during updates. Because weights cannot be subtracted in semigroups, deletions are especially critical as the affected generators have to be reset (set as ϕ).

These results on the dynamic case are later strengthened by Chazelle [22], and Brönnimann *et al.* [20], who prove that Fredman's results hold even without taking deletions into account (i.e. for sequences of insert and query operations). Furthermore, Brönnimann *et al.* improved this bound and extended it to $\mathbb{R}^d$. In $\mathbb{R}^d$, a conveniently chosen sequence of n insertions followed by n queries takes $n^{2-O(1/d)}$, in particular, $\Omega(n^{11/9}/\text{polylog}\ (n))$ for dimension two.

The extensions for the dynamic case are derived from the following results which define space-time tradeoffs for the static scenario. In this case, the model is restricted to a *faithful semigroup*[3].

Theorem 1 *(Chazelle) [22]: Simplex range searching on n points requires $\Omega(n/\sqrt{m})$ query time in two dimensions and $\Omega((n/\log n)/m^{1/d})$ query time in any dimension $d \geq 3$, where m denotes the amount of storage available. These bounds hold for a random point set, and therefore are valid in the worst case as well as on average.*

For the proof, the author uses, instead of simplices, the space between two parallel hyperplanes, namely *slabs*. However, such simplification does not extend to halfspaces since slabs are constrained to have a positive width. Later on, Brönnimann *et al.* overcame this issue and stated the following, somehow weaker (non-optimal), theorem for a lower bound on halfspace range searching.

Theorem 2 *(Brönnimann et al.) [20]: Halfspace range searching on n points, using m units of storage, requires a query time bounded by*

[3] A semigroup $(W, \oplus)$ is faithful if two linear forms (sums of variables) on W are identically equal only if the sets of variables involved are also equal. For example, $(\{0, 1\}, xor)$, where xor stands for exclusive or, is not faithful since $x\,xor\,x = y\,xor\,y$ for $x = 0, y = 1$. All the semigroups considered for range searching problems mentioned earlier are faithful. More details are given by Chazelle [22], who uses this restriction for the first time to prove lower bounds on range searching.

$$\Omega((\tfrac{n}{\log n})^{\frac{d^2+1}{d^2+d}}/m^{1/d})$$

The idea behind the proof of these theorems is based on the *characteristic graph* of the pair (P, R) of point set and range set considered. The characteristic graph is defined as $G = (V, X, E)$, where V is a set of variables, X is a set of generators that answer all possible queries, and E is the set of edges connecting variables to the generators where they appear. Each point has an associated variable where its weight is stored. The faithfulness of the underlying semigroup constrains the set of generators involved in the answer to a given query to contain exactly the set of variables associated to the points that match the query. The complexity of the range searching can be seen as the number of complete bipartite subgraphs in G combined with the size of these subgraphs. Intuitively, it is good to have generators for as large sets of variables as possible which, at the same time, answer as many queries as possible. The size of such complete bipartite subgraphs is bounded by the area of the smallest convex hull of random (under uniform distribution) points in $[0, 1]^d$. This is also related to a problem known as Heilbronn's problem[4].

Recently Arya, Malamatos, and Mount [11] have developed tighter lower bounds for halfspace range searching when the semigroup is *integral*. A semigroup $(W, \oplus)$ is integral if $x \oplus x \oplus \ldots \oplus x \neq x$, for all $x \in W$, with at least one application of $\oplus$. In contraposition to integral, there are *idempotent* semigroups. An idempotent semigroup is such that $x \oplus x = x$, for all $x \in W$. For example, $(\mathbb{R} \setminus \{0\}, +)$ is integral and $(2^P, \cup)$ is idempotent. There are semigroups such as $(\mathbb{C}, \times)$ which are neither integral nor idempotent. Obviously, for integral semigroups, the clusters that answer a query must be disjoint. This was used by Arya *et al.* to obtain slightly better lower bounds on range searching problems. One extra constraint they used was *convex clusters* (i.e. the convex hulls of the clusters do not contain any other point that is not part of the cluster). This restriction does not apply to all the data structures, for example, the optimal data structures proposed by Matoušek [73]. However, it can be adapted to other quasi-optimal data structures such as the one proposed by Chazelle, Sharir, and Welzl [36]. Arya *et al.* conjecture that the convex clusters constraint could be avoided. Their main result for exact halfspace range searching is given in the following theorem.

Theorem 3 *(Arya et al.) [11]: Halfspace range searching on n points with weights over an integral semigroup, using m units of storage, requires*

$$\Omega((\tfrac{n}{\log n})/m^{\frac{d+1}{d^2+1}})$$

for n sufficiently large.

Arya *et al.* [11] presented other important lower bounds on the arithmetic model that make distinctions on the type of semigroup as well.

2.2 The group model

This model is closely related to the arithmetic model except for the underlying semigroup which is, in fact, a group. This means that subtractions are allowed for the computation of the cumulative weight, making

[4]This classical problem of Heilbronn's [77] on a set P of n points in the unit square $[0, 1]^2$, ask for the asymptotic behavior of the supreme, over all possible point configurations, of the smallest area of a triangle with vertices in P.

the model far more powerful. Although most known algorithms for range searching follow the arithmetic model, there are problems for which subtractions allow more efficient solutions. One such example, as given by Chazelle and Rosenberg [32], is the problem of computing partial sums of weights associated to points contained in a given d-rectangle.

However, proving lower bounds under the group model seems to be a hard problem. One first step in this direction -and the only one we have knowledge of- has been presented by Chazelle [26] for the off-line case in two dimensions. He proved that for a set of n points, summing up the weight of those lying within n halfplanes takes $\Omega(n \log n)$ additions and subtractions. It seems this result can be further improved since the best know algorithm for this problem has time complexity close to $O(n^{4/3})$ [73].

2.3 The pointer machine model

The *pointer machine* model was defined by Tarjan [85]. Basically, a pointer machine consists of a finite (unbounded) memory, which is divided into units called registers. Each register may store a data value or a pointer. A data structure in the pointer machine is a collection of records with identical sets of labelled fields. Each field is stored in a data register or a pointer register, depending on the type of the field. An algorithm in the pointer machine is described as a finite set of labelled instructions. The instructions allowed are basically assignments among registers, arithmetic operations on data registers, and conditional jumps of the control to specific instructions.

Note that no pointer arithmetic is allowed in this model, which makes it simpler than the RAM model. So, to access a record in the data structure it is required that at least another record with a pointer containing its location has been previously visited. This defines a data structure as a directed graph with a *root* record which has in-degree equal to zero. Thus the space complexity of range searching on a pointer machine can be bounded by the minimum number of records needed in the data structure, and the time complexity, by the required number of visited records while answering a query. Under these assumptions, it has been shown that it is sufficient to consider only data structures that are rooted binary trees [23].

Chazelle and Rosenberg [33] derived lower bounds on this model for simplex range reporting using the principle of *filtering search* [21]. Filtering search is a way of amortizing the query time with the size of the query answer. In this case, it means that it is not necessary to locate the set of points to be reported very quickly if the size of the answer is large and the reporting process itself is slow. This basic idea allows smaller data structures and better space time tradeoffs than those previously presented in the arithmetic model. The following theorem formally states a suitable lower bound for simplex range reporting.

Theorem 4 *(Chazelle and Rosenberg) [33]: Simplex range reporting on a pointer machine requires* $\Omega(\frac{n^{1-\epsilon}}{m^{1/d}})$ *time for any fixed* $\epsilon > 0$*, where* n *is the number of points and* m *is the space used by the data structure.*

This bound allows query time $O((\log n)^b + r)$ with space $\Omega(n^{d-\epsilon})$ for any fixed $\epsilon > 0$. The authors conjectured that this polylogarithmic time bound could be also extended to halfspace range reporting with $\Omega(n^{\lfloor d/2 \rfloor - \epsilon})$ space.

2.4 The partition graph model

The partition graph model is a generalization of the idea behind the existing data structures for range searching. It defines a recursive decomposition of the primal and dual[5] spaces as an acyclic directed multigraph (*partition graph*) whose vertices define the nodes and leaves of a data structure. One node that has no incoming edges is distinguished as the *root* and serves as the starting point for range query algorithms. Associated to each node and leaf there is a region of the space and a bounded number of generators as in the semigroup model. Thus, the partition graph model inherits the properties of the arithmetic model, and combines it with the complexity of data structures (in a way similar to the pointer machine model) in order to obtain better lower bounds.

Erickson [47] presented this model as a tool for proving new lower bounds on hyperplane and halfspace queries. The main result from Erickson's paper is stated in the following theorem.

Theorem 5 *(Erickson) [47]: Halfspace emptiness on the partition graph model, with regions of bounded complexity, requires*

$$\Omega((\tfrac{n}{\log n})^{\frac{\delta^2+1}{\delta^2+\delta}}/m^{1/\delta})$$

query time, where $d \geq \delta(\delta+3)/2$, n is the number of points and m is the space used by the data structure.

The similarity between this result and the lower bound for halfspace range searching as in Theorem 2 is not a coincidence: the former has been derived from the latter. The derivation of this bound is based on three main elements. First, a partition graph that supports hyperplane emptiness queries allows the construction of a generator scheme that answers halfspace queries in the arithmetic model with the same complexity. So, the bound derived in Theorem 2 automatically becomes a bound for the hyperplane emptiness problem. Second, a hyperplane emptiness problem can be transformed into a halfspace emptiness one in a higher-dimensional space. This is done by converting points and hyperplanes in $\mathbb{R}^d$ expressed in homogeneous coordinates into points and hyperplanes in $\mathbb{R}^{d(d+3)/2}$ also in homogeneous coordinates using the transformation

$$\sigma_d(x_0, x_1, \ldots, x_d) = (x_0^2, x_1^2, \ldots, x_d^2, \sqrt{2}x_0x_1, \sqrt{2}x_0x_2, \ldots, \sqrt{2}x_{d-1}x_d),$$

with the property of preserving point hyperplane incidences, except that when a point p is not on a hyperplane h, no matter whether it is above or below h, $\sigma(p)$ always appears above $\sigma(h)$. Finally, given a partition graph with regions of bounded complexity in $\mathbb{R}^{d(d+3)/2}$ that answers halfspace emptiness queries, another partition graph, also with regions of bounded complexity in $\mathbb{R}^d$, can be constructed for halfspace emptiness.

The previous bound is the first nontrivial one for halfspace emptiness. However, the existing gap between this bound and the best known algorithm -as it will be shown in section 3- suggests that it can be further improved. One possible improvement may come as consequence of better lower bounds for halfspace range searching on the arithmetic model. Another possibility is finding a better transformation to convert a halfspace emptiness problem into a hyperplane emptiness problem, hopefully $\sigma : \mathbb{R}^d \rightarrow \mathbb{R}^{2d}$.

[5] A definition of dual space appears in section 3.1 under *Duality*. The reader is also referred to [79] (section 6.5) for a more extensive definition.

Range	Problem	Space	Query time	Source
halfspace	general	m	$\frac{n}{m^{1/d}}$	[73]
halfspace	reporting, $d = 2$	n	$\log n + k$	[37]
halfspace	reporting, $d = 3$	$n \log n$	$\log n + k$	[6]
halfspace	reporting, $d > 3$	$n \log \log n$	$n^{1-1/\lfloor d/2 \rfloor} \text{polylog } n + k$	[73]
halfspace	emptiness, $d = 2$	n	$\log n$	[80]
halfspace	emptiness, $d = 3$	n	$\log n$	[43]
halfspace	emptiness, $d > 3$	$n \log \log n$	$n^{1-1/\lfloor d/2 \rfloor} 2^{c \log^* n}$	[73]
simplex	general, $d = 2$	n	$n^{1/2} \log n$	[34]
simplex	general, $d = 3$	$n \log n$	$n^{2/3} \log^2 n$	[34]
simplex	general, $d \geq 2$	m	$\frac{n}{m^{1/d}} \log^{d+1} \frac{m}{n}$	[73]

Table 2: Upper bounds for different versions of on-line range searching problem.

2.5 Open problems

In most variants of non-orthogonal range searching there are still at least polylogarithmic gaps between the lower bounds and the complexity of available algorithms, take simplex range searching for example. Therefore, a whole family of open problems consists in closing the polylogarithmic gaps. The following is a summary of other open problems, some of which have already been mentioned above.

- Can the lower bounds for halfspace range searching on the arithmetic model be improved such that it becomes closer to the lower bound for simplex range searching?
- Can better lower bounds for dynamic data structures be found? These bounds should include the complexity of update operations as well as space, query time, and preprocessing time complexities.
- Can better lower bounds on the group model be found?
- Generalize the bounds on the integral semigroup to ranges that are not necessarily convex.
- Bridge the existing gap between lower bounds and algorithms for halfspace emptiness searching.

3 Range searching algorithms and data structures

As shown before in terms of lower bounds, it is very unlikely -not possible under the computational models considered- that a data structure of linear size can answer non-orthogonal range queries in polylogarithmic time. Therefore, the available data structures usually appear classified into two main classes: those that aim to answer queries in polylogarithmic time using superlinear storage, and those that rely on a linear size data structure yet allowing queries in sublinear time. We will review the most significant of them as well as other solutions that encompass the whole spectrum of data structures in between. Most of the best results to date are summarized on table 2.

3.1 Geometric preliminaries

Here we define some terms that will be used during the presentation of different range searching data structures. The reader might want to skip this section and come back here whenever new terms come along. Note that, for the sake of simplicity, we have not been rigorous in the description of ranges or regions since we do not refer to them as open or closed sets. The reader should also assume that the input data contain no degeneracies. Although most degeneracies can be handled in a systematic way, they require special treatment that may obscure the explanation of the reviewed results. Thus, constraints imposed on the input data include *general position*[6].

- *Duality:* Duality is one of the most largely used tools in Computational Geometry. It defines mapping between points and hyperplanes in $\mathbb{R}^d$. A duality mapping function uniquely transforms point coordinates into hyperplane parameters and vice versa. One such example is $D : (p_1, p_2, \ldots, p_d) \Leftrightarrow p_d + x_d = p_1x_1 + p_2x_2 + \ldots + p_{d-1}x_{d-1}$. This mapping has the properties of preserving incidence and relative position, that is, given a point p and a hyperplane h, if p is on, above, or below h, then the point h', dual of h, will be on, above, or below the hyperplane p', dual of p, respectively. See Figure 6 for a 2-dimensional example.

- *Ham-sandwich theorem:* Using a single straight cut, any ham sandwich can be cut into two halves such that each half contains the same amount of ham, cheese and bread. A discrete version of this theorem states that d finite sets of points in $\mathbb{R}^d$ can be all simultaneously bisected by a single hyperplane. Edelsbrunner [44](§4.2) formally states the ham-sandwich theorem including a proof and other related results.

- *VC-dimension:* Consider the range space (P, R), $Q \subseteq P$, and the set $\Pi_R(Q) = \{Q \cap r | r \in R\}$. We say that R *shatters* Q if $\Pi_R(Q)$ is equal to the power set of Q (the set of all subsets of Q). The Vapnik-Chervonenkis-dimension (VC-dimension) of (P, R) is the size of the largest Q that is shattered by R. For example, the range spaces of the form $(\mathbb{R}^d, H_d)$, where H_d is the set of d-dimensional halfspaces has VC-dimension $d + 1$. In particular, for $(\mathbb{R}^2, H_2)$, we can easily check that there are no 4 points for which all subsets can be obtained as the intersection with a halfplane; however, this is possible for 3 points. Therefore, the VC-dimension of $(\mathbb{R}^2, H_2)$ is 3. If the size of the subset Q that can be shattered by R is unbounded then we say that the VC-dimension is infinite. The concept of VC-dimension was first introduced by the authors that give it its name [86].

- *ϵ-net:* Consider the range space (P, R), and $0 < \epsilon < 1$ a constant. An ϵ-net of P (with respect to R) is a sample set $N \subseteq P$ such that for any $Q \in \{P \cap r, r \in R\}$, if Q is relatively large ($|Q|/|P| \geq \epsilon$), then it contains at least one element of N. For example, let (C, H_2) be a range space where C is a set of points on a circle and H_2 the set of all halfplanes. A (1/4)-net is realized by the points in positions 0, $\lceil n/4 \rceil$, $\lceil n/2 \rceil$, and $\lceil 3n/4 \rceil$ following a circular order of the points around the center of the circle. It is obvious that every halfplane that contains more than $\lceil n/4 \rceil$ points will contain at least one from the (1/4)-net. The concept of ϵ-nets was introduced by Haussler and Welzl [61] and presently has many applications in Computational Geometry.

[6] We say that a set of points in $\mathbb{R}^d$ is in general position if no more than k points lie on the same $k-1$-dimensional hyperplane, for $1 \leq k \leq d$. For example, in a 3-dimensional space, a set of points is in general position if no 4 points lie on the same plane, no 3 points are on the same line, and no 2 points coincide. We also restrict the set to no 5 points on the same (3-dimensional) sphere, no 4 points on the same 2-dimensional sphere (circle), or, in general, no $k+2$ points on the same k-dimensional hypersphere, for k as described above. Note that the latter restriction can be neglected in some cases. Besides, a set of $d-1$-hyperplanes is in general position if its dual set of points is in general position, no two hyperplanes are parallel, and no one is *vertical* (parallel to axis coordinate d of the space).

- *ϵ-cutting:* Let H be a set of m hyperplanes in $\mathbb{R}^d$, and $0 < \epsilon < 1$ a constant. An ϵ-cutting for H is a partition of $\mathbb{R}^d$ into simplices, such that every simplex is intersected by at most ϵm hyperplanes of H. A weighted version of ϵ-cuttings is defined over a set of weighted hyperplanes, according to certain function $w : H \to R$. Thus, in the weighted version we say that a simplicial partition of $\mathbb{R}^d$ is an ϵ-cutting if the total weight of the hyperplanes intersecting each simplex is at most $\epsilon w(H)$, where $w(H)$ is the sum of all hyperplane weights. Chazelle, Friedman, and Matoušek [28, 69] used early versions of ϵ-cuttings, but the concept was later formalized by Matoušek [70].

3.2 Deterministic partition trees

There are simple linear-size data structures that allow sublinear time orthogonal range searching. Two examples are *quad trees* [52], and *K-d trees* [16]. A generalization of these was first developed by Willard [88] for halfplane range searching. A data structure that follows Willard's description, is known as a *partition tree*. He proved that a set of J lines (and semilines[7]), not necessarily parallel to the axis coordinates, exists such that it partitions the plane into $2J$ regions, each region contains the same number of points of P, and at most $J+1$ regions can be intersected by any halfplane. The recursive application of this partitioning scheme leads to a partition tree. This is applied until each region contains a small constant number of points. A node v of the tree is created for each region and the total weight w_v (in the semigroup sense) of the elements in the corresponding region is stored in it. The root of the tree is associated to the entire space. Every non-leaf node, with region *Reg*, has as children the nodes that contain the regions resulting from partitioning *Reg*. To answer a query based on this type of partition trees, and other variants that will be seen later on, the following algorithm is applied.

1. Initialize the variable ANS (for the answer) as null and make v equal to the root of the tree.
2. If the region *Reg* associated to v is a subset of the query range r, then make ANS = ANS + w_v; otherwise, if $Reg \cap r = \emptyset$, leave ANS unchanged, otherwise, repeat this step for $v = v_1, v_2, \ldots, v_k$, where each v_i is a child of v, $i = 1, \ldots, k$.

Following the algorithm just described, the query time on Willard's partition trees is minimum when $J = 3$ ($O(n^{0.774})$). An example for $J = 3$ appears in Figure 1.

A generalization to this partitioning idea when $J = 2$ is the ham-sandwich cut theorem in two dimensions. Based on this theorem, Edelsbrunner and Welzl [45] developed a similar version of partition tree with a more efficient partitioning scheme. In this case, two adjacent regions at level $l-1$ are divided into four regions at level l, that contain the same number of points, as shown in figure 2. This partitioning scheme leads to $O(n^{0.695})$ query time. The construction of the partition tree can be done in $O(n \log n)$ using an algorithm by Lo, Matoušek, and Steiger [68] that finds a ham-sandwich cut in linear time.

In $\mathbb{R}^3$ a similar partition scheme exists: a set of points can be partitioned into 8 equal parts by 3 planes. Yao *et al.* [91] proposed a partition tree based on this partition scheme. An appropriate grouping of the regions in the partitioning scheme leads to a data structure with query time $O(n^{0.899})$. It would be desirable to further extend this idea of partitioning a set of points P in $\mathbb{R}^d$ into 2^d equal parts using d hyperplanes. This

[7] We refer to a semiline as a line that is infinite in only one direction.

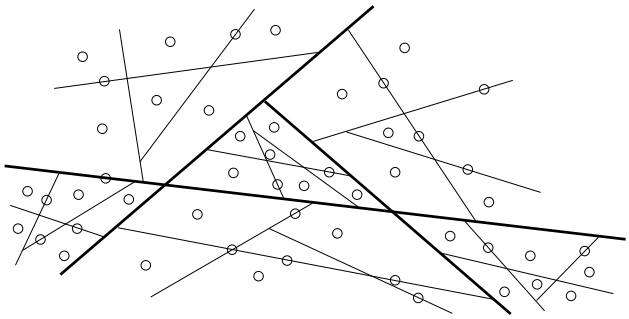

Figure 1: Two levels of a partition tree as defined by Willard [88] with $J = 3$.

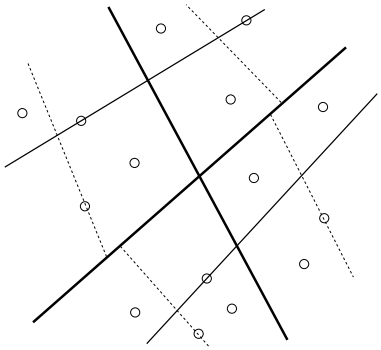

Figure 2: In this partition scheme, 2 adjacent regions in one level are bisected by one line such that 4 new regions are formed with similar number of points. We show partitioning lines at different levels in the tree with different thickness and styles.

question is open for $\mathbb{R}^4$ and unfortunately, not possible for $d \geq 5$, as shown by Willard [88]. However, Yao and Yao [90] derived a partitioning scheme in $\mathbb{R}^d$ with possibly more than d hyperplanes or half-hyperplanes that share a common (center) point and have the property of partitioning P into 2^d equal parts such that at most $2^d - 1$ of them are intersected by a query hyperplane. Figure 3 shows an example of this partition scheme for $d = 3$. This scheme allows $O(n^{\log_2(2^d-1)/d})$ query time, which is still a small improvement over linear search.

3.3 Randomized preprocessing

An important event in Computational Geometry was the adoption of randomized algorithms in the middle of the 80's. Seminal papers by Haussler and Welzl, and by Clarkson introduced random samplings as a technique to construct data structures for range searching. Clarkson's results [40] will be reviewed later in section 3.8 as we see efficient data structures for halfspace reporting queries.

Haussler and Welzl [61] introduced the idea of *epsilon*-nets and showed how to use them on range spaces of finite VC-dimension for the construction of partition trees. They proved that a sufficiently large random sample of the input points P is an ϵ-net with high probability. The size of the ϵ-net is solely determined

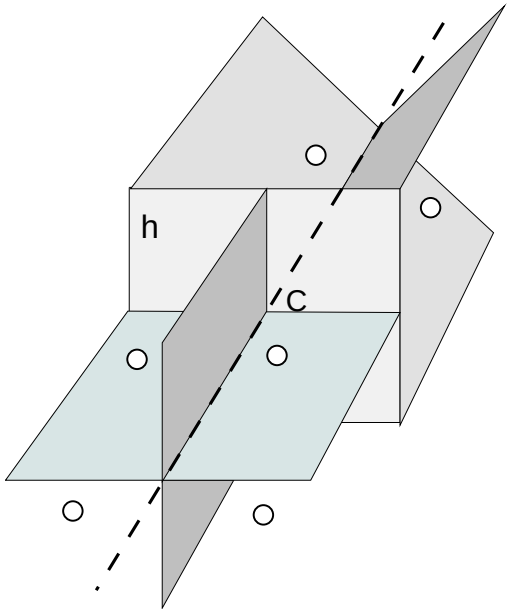

Figure 3: Example of the partitioning approach by Yao and Yao [90] for $d = 3$. The partition is realized by different pairs of halfplanes on each side of plane h. A query hyperplane will intersect at most 7 out of 8 regions.

by the VC-dimension and not input the. Every combination of d of these points determines a hyperplane in d dimensions, and the arrangement realized by all such hyperplanes can be used as a partition of the space. They also proved that by using this partitioning scheme, the number of points contained in the regions intersected by any query hyperplane is at most ϵn. Thus, applying this scheme recursively leads to a partition tree with $O(n^{\frac{d(d-1)}{d(d-1)+1}+\gamma})$ query time, where $\gamma > 0$ is a parameter that determines the value of ϵ and therefore the (constant) size of the ϵ-net. For $d \geq 2$ this value substantially improves the query times seen above for deterministic partition trees.

Matoušek and Agarwal [69, 1] were the first to provide deterministic algorithms to obtain ϵ-nets in $\mathbb{R}^2$ that leads to the construction of a partition tree in $O(n \log n)$ time like in the randomized approach. A generalization of ϵ-nets when the sample set is not a subset of the input points is known as a *weak ϵ-net*. There are also algorithms for constructing weak ϵ-nets, some of them for more general ranges spaces, such as $(\mathbb{R}^d, \mathcal{C})$ where $\mathcal{C}$ is the set of all convex ranges (with infinite VC-dimension). Matoušek and Wagner [76] provide an algorithm (and consequently upper bounds) for weak ϵ-nets on convex ranges. Aronov *et al.* [10], and Babazadeh and Zarrabi-Zadeh [12] provide bounds on ϵ for weak ϵ-nets of small constant size. These bounds, in turn, provide bounds for the constants involved in the complexity of range searching schemes based on weak ϵ-nets.

3.4 Spanning trees

A more efficient type of partition trees introduced by Welzl [87] allows the regions in the partition scheme to overlap. This idea is better seen on spanning trees on the input points. In order to explain the relationship between spanning trees and partition trees, we should first introduce some definitions.

Given a range space (P, R), we say $r \in R$ *crosses* the edge defined by $p_0, p_1 \in P$ if $|r \cap \{p_0, p_1\}| = 1$. The *crossing number* of r in a spanning tree $\mathcal{T}$ is the number of edges of $\mathcal{T}$ crossed by r. Then, the crossing number of $\mathcal{T}$ ($\sigma(\mathcal{T})$)is the maximum crossing number over all $r \in R$. An *optimal spanning tree* of P is one that minimizes the crossing number. The maximum number of visited nodes in a partition tree to answer any query range is defined as *visiting number*.

We can now see how the visiting number of a partition tree T is bounded by the crossing number of a corresponding spanning tree. In fact, for any spanning tree $\mathcal{T}$ there is a spanning path $\mathcal{P}$ on the same set of points with $\sigma(\mathcal{P}) \in \Theta(\sigma(\mathcal{T}))$. Such a spanning path can be built as the sequence of points along a preorder traversal of the spanning tree. Figure 4 (a) shows a spanning tree and the corresponding spanning path for a sample set of points. A lemma by Welzl establishes the relationship between the visiting number of a partition tree and the crossing number of a corresponding spanning path.

Lemma 6 *(Welzl) [87]: Let (P, R) be a range space and N a set of n points in P.*

(i) if T is a partition tree of N, then there exists a spanning path $\mathcal{P}$ of N such that $\sigma(\mathcal{P})$ is at most twice the visiting number of T.

(ii) if $\mathcal{P}$ is a spanning path of N, then there exists a partition tree T of N such that the visiting number of T is at most $2 \cdot \sigma(\mathcal{P})\lceil \log n \rceil + 1$.

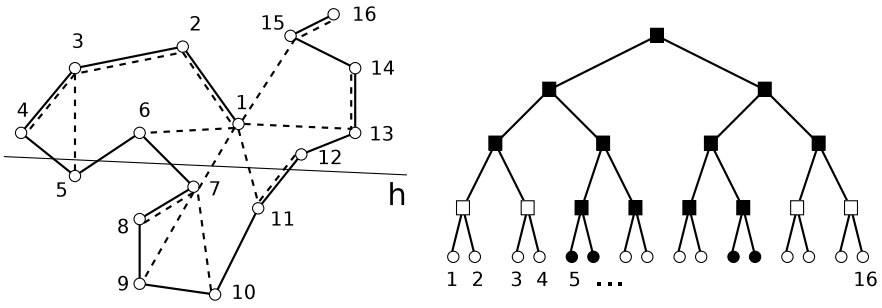

Figure 4: (Adapted from [27](Figure 5.8)). Left: spanning tree for a set of points in the plane (dashed lines), associated spanning path (solid lines), and query line h that defines a query halfplane. Right: corresponding partition tree. The nodes coloured in black are those visited while answering the query for the halfplane below h.

As a consequence of lemma 6, an upper bound on the crossing number for an optimal spanning tree of a set of n points also provides an upper bound on the visiting number of an optimal partition tree. An upper bound proved by Welzl states that $\sigma(\mathcal{P}) \in O(n^{1-1/\delta} \log n)$, where δ is a constant that only depends on the

dimension of the space, and the shape of the ranges. More specifically, the number of cells in an arrangement defined by m ranges from R is $O(m^\delta)$. The value δ is related to the VC-dimension and coincides with d for R the set of halfspaces and balls in $\mathbb{R}^d$. An improvement to the previous bound was derived by Chazelle and Welzl [34]: they proved that $\sigma(\mathcal{P}) \in O(n^{1-1/d})$.

It follows that spanning paths with low crossing number define nice data structures for range searching. If one is provided with an efficient algorithm to determine the edges crossed by a query range, then the query is reduced to a set of one-dimensional range queries. This is possible because the points are sorted along the spanning path. Each one of these one-dimensional problems is equivalent to a partial sum problem which has a known efficient solution. According to Yao's results [89], the latter can be done in $O(\alpha(n))$ time, where $\alpha(n)$ is the inverse of Ackermann's function[8]. So, in the overall, the query time complexity would be $O(n^{1-1/\delta}\alpha(n))$. However, the problem remains for determining the set of crossed edges. This can be done efficiently in dimensions 2 and 3, but does not generalize to higher dimensions. For example, Chazelle [27](§5.3) shows how to answer triangle range searching in the plane in $O(\sqrt{n}\log n)$ time using $O(n)$ space. Note that this algorithm is quasi-optimal compared to the lower bound given in Theorem 1.

3.5 Simplicial partitions

A generalization of spanning trees with low crossing number was given by Matoušek [71]. He considers partitioning schemes where the regions are also not necessarily disjoint; however, in his approach the number of regions is not bounded by a constant. Previously, Alon *et al.* [8] had proved that for some range space there is no efficient partition tree with a constant-bounded number of regions. Matoušek's approach is based on *simplicial partitions*. A simplicial partition on a set of points P in $\mathbb{R}^d$ is a collection of pairs of simplices and subsets of P ($\Pi = \{(P_1, \Delta_1), \ldots, (P_m, \Delta_m)\}$), such that the sets P_i (*classes*) form a partition of P and each Δ_i is a simplex in $\mathbb{R}^d$ containing P_i (see Figure 5).

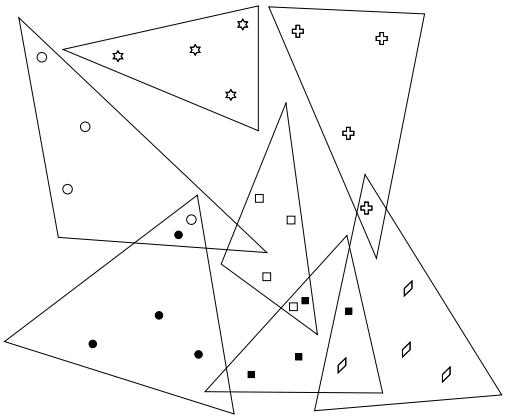

Figure 5: (Adapted from [74](Figure 3)) Simplicial partition in $\mathbb{R}^2$. Different symbols are used for points in different classes.

We say that a hyperplane *crosses* a simplex if the interior of the simplex intersects the hyperplane. So, we can define the crossing number of a simplicial partition similar to the crossing number of a spanning tree,

[8] Ackermann's function is a recursive function that grows extremely fast with respect to n. Therefore, its inverse ($\alpha(n)$) grows very slowly, even slower than logarithmic functions.

that is, the maximum number of simplices stabbed by a hyperplane. We are interested in finding a simplicial partition Π with low crossing number. The construction of Π is based on the weighted version of ϵ-cuttings. Let r be a parameter, $1 < r \leq n/2$, whose value will be better specified later on. Π is constructed in a way that satisfies $n/r \leq |P_i| \leq 2n/r$. The first step of the algorithm is to find a set of sample hyperplanes H (test set) of size at most r that bounds the crossing number of any simplicial partition as described above, for hyperplanes in general. H can be constructed using duality and ϵ-cuttings[9]. Provided such a test set, the second step is to construct the simplices of Π one at a time. Thus, at time i the simplices $\Delta_1 \ldots \Delta_i$ have already been built and the hyperplanes of H are assigned weights depending on the number of intersections with respect to $\Delta_1 \ldots \Delta_i$. Then, we find a ϵ_i-cutting (Ξ_i) for the weighted hyperplanes of H such that one of the simplices in Ξ_i contains enough points to be selected as Δ_{i+1} on the simplicial partition. The value of ϵ_i depends on the number of remaining points to be covered by Π. This construction proves the following theorem.

Theorem 7 *(Matoušek) [74]: Let P and r be as defined above. Then, there exist a simplicial partition for P with $O(r)$ classes, such that $n/r \leq |P_i| \leq 2n/r$ for every class P_i, with crossing number $O(r^{1-1/d})$.*

A partition tree built upon simplicial partitions supports halfspace and simplex range queries in time $T(n) = O(r) + O(r^{1-1/d})T(2n/r)$. Now we should give an appropriate value for r. For $r = n^{1-1/d}$, a data structure of linear size can be constructed in $O(n \log n)$ time, allowing query times $O(n^{1-1/d} \log^{O(1)} n)$. This solution is optimal up to a polylogarithmic factor if compared to the corresponding lower bound. However, it is possible to get rid of the $\log^{O(1)} n$ factor in the query time. This was achieved by Matoušek [73], using Chazelle's *hierarchical cuttings*[10] [25] (see Theorem 8).

3.6 Polylogarithmic query time

All the data structures seen above use $O(n)$ space and allow sublinear query time. We will now examine other data structures that make use of larger space in order to reduce query times to polylogarithmic. One way to achieve this goal is to precompute all possible answers for the range space of interest and build an index on the answers such that they can be efficiently located.

For halfspace range searching, one can compute all the possible answers using duality. Suppose we are only interested in halfspaces that lie above the defining query hyperplanes -for halfspaces below the defining hyperplanes we can procede symmetrically. Let (P, H^+) be such range space. Using the duality mapping $D : (p_1, p_2, \ldots, p_d) \Leftrightarrow p_d + x_d = p_1 x_1 + p_2 x_2 + \ldots + p_{d-1} x_{d-1}$, a point p is above a hyperplane h, if and only if its dual hyperplane p' falls below the dual point h' (see Figure 6). P' denotes the set of hyperplanes dual to the points in P. The answer to a range $h^+ \in H^+$ (with bounding hyperplane h) consist of the subset of P' below h' in the dual space. One can use arrangements for this purpose. The arrangement of hyperplanes $A(P')$ divides the dual space into convex cells, such that every cell realizes a set of points with identical answer. Thus, in order to answer the original query we only need to precompute the subset of P' lying below each cell and use *point location* techniques to efficiently find the cell in which h' lies. The preprocessing can be done in $O(n^{d+1})$ time using a naive approach (testing each hyperplane against each

[9] One such test set H is given by the dual of the set of vertices of a $(1/r)$-cutting for the set of hyperplanes dual of the input set of points.

[10] A hierarchical cutting is an ϵ-cutting that is built globally for all resolutions of the application of the partitioning scheme.

cell). This can be improved to $O(n^d)$ as presented by Edelsbrunner [44]. The second step, point location, can be done efficiently in 2 dimensions for any convex cell subdivision using monotone subdivisions, for instance. In higher dimensions, efficient point location on arrangements of hyperplanes can be achieved due to data structures based on ϵ-cuttings. Here "efficient" means $O(\log n)$ query time and $O(n^d)$ space. An even more efficient technique to solve halfspace range searching uses the following result on hierarchical cuttings.

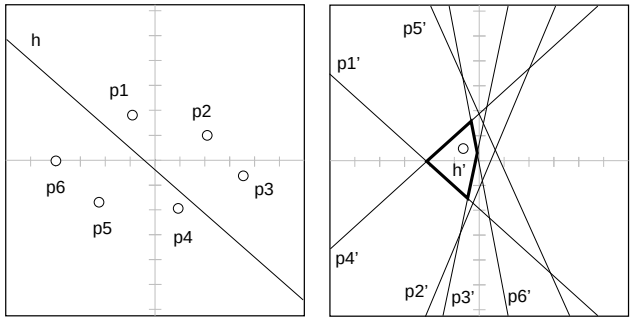

Figure 6: Left: primal space. Right: dual space. The thick line polygon depicts the boundary of the cell that contains h' in the dual space.

Theorem 8 *(Chazelle) [25]: Let H be a set of n hyperplanes, $r \leq n$ a parameter, and $k = \lceil \log_2 r \rceil$. There exist $\Xi_1, \ldots, \Xi_k$, where 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, $\Xi_1, \ldots, \Xi_k$ can be computed in time $O(nr^{d-1})$.*

This theorem can be directly applied to the construction of a hierarchical data structure for divide-and-conquer on the set of dual hyperplanes. Matoušek [73] proved that this data structure can be built with $O((n/\log n)^d)$ space allowing $O(\log n)$ query time.

For simplex range searching, however, precomputing all possible answers is very expensive in terms of storage. There are $O(n^{d(d+1)})$ possible answers for simplex range searching in $\mathbb{R}^d$, which leads to $\Omega(n^{d(d+1)})$ storage using this approach. Cole and Yap [42] gave a solution to the planar problem based on a partitioning scheme that decomposes the original query triangle into smaller ones. They proved that their algorithm is near-optimal, with $O(\log n \log\log n)$ query time and $O(n^2/\log n)$ space. Later on, Chazelle *et al.* [36] gave a quasi-optimal solution for general dimension d with $O(n^{d+\epsilon})$ space and $O(\log n)$ query time. His approach relies on a *multilevel data structure* that is based on halfspace range searching techniques (see section 3.9). Matoušek [73] shows how the space bound can be improved to $O(n^d)$ at the expense of an increase on the query time bound to $O(\log^{d+1} n)$.

3.7 Space-time tradeoffs

So far we have reviewed linear size data structures that allow range searching in sublinear time (sections 3.2 through 3.5), as well as larger data structures that allow logarithmic query time (section 3.6); however, we may be interested in an intermediate solution that better fits the problem and the computational resources available (computer memory, processing speed, etc.). In other words, we need a tradeoff between space and

query time. This can be easily done by combining two types of hierarchical data structures: one of linear size, and one of logarithmic query time. Agarwal and Erickson [2] explain how to accomplish this. The rationale of the construction is explained in the following (see Figure 7).

Let $D_1(P)$ and $D_2(P)$ be two hierarchical range searching data structures on the set of points P for some given range space, where $D_1(P)$ allows logarithmic query time and D_2 is of linear size, both efficient. We start the construction of a new data structure D by pruning the subtrees of $D_1(P)$ on all nodes representing clusters of relatively small size -small compared to certain parameter τ that depends on the space dimension, the input size, and the desired size of the data structure. Then, for each node i, with corresponding cluster N_i we attach a data structure $D_2(N_i)$ as a child of i. It has been proved that by doing so, the data structure continues to be efficient and the storage can be limited to a size m, as desired. In other words, given a target size m for the data structure, we compute τ to decide on what nodes of $D_1(P)$ to prune and proceed as explained above to obtain D. D will answer range searching as efficiently as possible. For halfspace range searching, for instance, the queries can be answered in time $O(n/m^{1/d} + \log(m/n))$, which is near-optimal compared to the lower bounds presented in Theorem 1. The technique just presented can be applied to more general data structures $D_1(P)$ and $D_2(P)$. Agarwal and Erickson give more details about this [2].

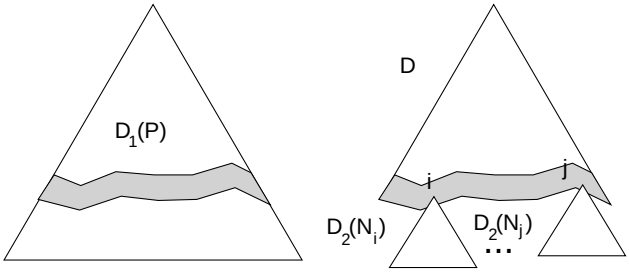

Figure 7: (Adapted from [2](Figure 7)). Left: A data structure of type D_1. Right: Final data structure D consisting of a pruned version of $D_1(P)$ with trees $D_2(N_i)$ and $D_2(N_j)$ attached to nodes i and j of $D_1(P)$.

3.8 Special cases

One of the most surprising results in non-orthogonal range searching is related to halfspace reporting queries. A halfspace reporting query, where the answer has size k, can be answered in $O(\log n+k)$ time using linear size data structures for 2 dimensions and near-linear size data structures for 3 dimensions, as shown by Chazelle *et al.* [37] and Chazelle and Preparata [31], respectively. The 3-dimensional case was later improved by Aggarwal *et al.* [6]. This may, at first, seem contradictory compared to the lower bounds obtained for halfspace range searching in the semigroup model. However, it can be understood in the context of *filtering search*. As informally presented in section 2.3, filtering search describes a way of amortizing the query time by using the size of the output, given that a large answer requires a slow reporting phase, thus, need no optimal seek time. Extensions to higher dimensions and small improvements for the 3-dimensional case were provided by Clarkson [41] based on combinatorial results on *j-sets*[11]. It is of combinatorial interest to know the total number of j-set for any input set of points. In the data structures presented by Chazelle and Preparata [31], and Clarkson [41], the total number of i-sets, for all $i \leq j$ is what is accounted for.

[11] Given a set of points P, a j-set J is a subset of P of size j such that $J = P \cap h$ for some halfspace h. In simple words, a j-set is a possible output for a range reporting query.

Clarkson's data structure allows $O(\log n + k)$ query time, takes $O(n^{\lfloor d/2 \rfloor + \epsilon})$ space, and can be built in $O(n^{\lfloor d/2 \rfloor + \epsilon})$ expected time using a randomized approach. Matoušek [72] presented another data structure for this problem based on cuttings resulting in $O(n \log\log n)$ space and $O(n^{1-1/\lfloor d/2 \rfloor} \log^{O(1)} n + k)$ query time. Clarkson's and Matoušek's techniques combined provide slightly better space-time tradeoffs.

Halfspace emptiness is related to halfspace range reporting: a halfspace emptiness query can be answered using a range reporting data structure just by excluding the output reporting step ($O(k)$). However, some of these data structures can be tailored for halfspace emptiness queries with better results. An efficient data structure for dimension 3 has been presented by Dobkin *et al.* [43] with $O(\log n)$ query time using $O(n)$ space. For higher dimensions, a variant of Matoušek's data structure answers emptiness queries in $O(n^{1-1/\lfloor d/2 \rfloor} 2^{O(1)\log^* n})$ time and $O(n)$ space, at the expense of more preprocessing. Another data structure for halfspace emptiness by Matoušek and Schwarzkopf [75] is comparable to Clarkson's, with $O(n^{\lfloor d/2 \rfloor} \log^{O(1)} n)$ space and $O(\log n)$ query time.

3.9 Multilevel data structures

For certain applications, one may want to perform range queries defined by more complex range shapes. This can be done in a non-trivial way through *multilevel data structures* whenever the query shape can be defined in terms of simpler shapes. Notice that for a query shape defined as the union of simpler query shapes, one perform a range query for each simpler query shape and then combine all the solutions. Therefore, multilevel data structures are meant for a query shape q that is the intersection of $q_1, \ldots, q_k$ simpler ranges, provided that there exists a data structure to answer each of the q_i. Thus, by conveniently combining the data structures for all q_i, we obtain a multilevel data structure that can answer q efficiently. This technique was first used by Bentley [17, 18] in a data structure called *range trees*. Range trees were designed for orthogonal range searching, however, they generalize to other types of range spaces. We will now proceed with a review of range trees.

Let (P, R) be a range space, $P \subseteq \mathbb{R}$, $R = R_1 \times R_2$, where R_1 and R_2 are intervals in $\mathbb{R}$. $BST_d(N)$ denotes a binary search tree on $N \subseteq P$, where the points in N are sorted according to the d-coordinate ($d \in \{1, 2\}$). Clearly, an instance of $BST_1(P)$ can solve 1-dimensional range queries on the first coordinate, provided that each node i of the tree contains the generator (partial sum) corresponding to the set of points (N_i) of the spanned interval. We can also store a binary search tree $BST_2(N_i)$ on each node i such that another 1-dimensional range search can be performed on the subset N_i according to the second coordinate. The resulting data structure T will allow answering orthogonal range queries as follows. Given a query rectangle $R = R_1 \times R_2$, we locate all the nodes of T associated to the clusters that realize the answer of R_1 in $O(\log n)$ time. Then, we can perform a range query according to R_2 on those nodes in $O(\log n)$ time per node. So, with overall $O(\log^2 n)$ time, orthogonal queries can be answered in 2 dimensions using $O(n \log n)$ space. In this case, the query time can be reduced to $O(\log n)$ using *fractional cascading*[12].

As mentioned above, the idea of combining hierarchical data structures for searching according to multiple criteria can be generalized to any type of search in a similar way: a node of the data hierarchy at level l contains a level $l + 1$ data structure for its associated cluster (subset of points). The description of the

[12] Fractional cascading is a technique used to speedup searches at the cost of a small increase of the data structures size. This is achieved by linking different parts of the data structure as a postprocessing step, such that searching in one part makes easier subsequent searches on other parts. A general framework for fractional cascading was provided by Chazelle and Guibas [29, 30]. They provide several examples and applications.

ranges must be finite so, the number of nesting levels is bounded by a constant. For instance, simplex range searching can be decomposed as a set of halfspace searches (one per bounding hyperplanes). In general, multilevel data structures, multiply the query time and space by a factor of $O(n^\epsilon)$ ($\epsilon > 0$)per level. Thus, for the case of simplex range searching, data structures that are directly based on a simplex partitioning scheme are more efficient. For example, it is better to use the simplicial partitions seen earlier [73] rather than a multilevel data structure resulting from the combination of halfspace range searching data structures.

3.10 Semialgebraic ranges

So far we have reviewed range searching with relatively simple range shapes (halfspaces and simplices). However, at the beginning of this paper we saw an example that involved ball range searching. We will now see how to deal with range searching in the presence of balls, or more general, range shapes. We consider ranges defined as *Tarski cells*. A Tarski cell is a region of $\mathbb{R}^d$ delimited by a polynomial of bounded degree in d variates. The intersection of a bounded number of Tarski cells is also a Tarski cell. Thus, Tarski cells are more general ranges of bounded complexity. This more general setting is known as *semialgebraic* range searching.

Yao and Yao [90] showed how a range space $(\mathbb{R}^d, T)$ can be transformed into another range space $(\mathbb{R}^\delta, H_\delta)$ preserving containment relationships, where T is a set of polynomial inequalities, $\delta > d$, and H_δ is the set of halfspaces in $\mathbb{R}^\delta$. This technique is known as *linearization*. Consequently, range searching on Tarski cell ranges can be answered by means of halfspace range searching in higher dimensions. Tarski cells delimited by the intersection of more than one polynomial can be handled by multilevel data structures as explained in section 3.9. A direct application of the method proposed by Yao and Yao does not give the smallest value of δ, which is indeed desirable. Agarwal and Matoušek [3] give an algorithm to determine the transformation that leads to the smallest δ value.

The following is an example of how disk range searching in 2 dimensions can be optimally transformed into a 3-dimensional halfspace range searching. Let $(x_1 - a_1)^2 + (x_2 - a_2)^2 \leq a_3$ be a disk with center (a_1, a_2) and radius a_3. The transformation consists of *lifting* the points of the form (x_1, x_2) onto the 3-dimensional parabolloid $(x_1, x_2, {x_1}^2 + {x_1}^2)$, and transforming the query disk defined by (a_1, a_2, a_3) to the halfspace $\{y \in \mathbb{R}^3 : {a_3}^2 - {a_1}^2 - {a_2}^2 + 2a_1y_1 + 2a_2y_2 - y_3 \geq 0\}$. Similarly ball range searching in $\mathbb{R}^d$ can be transformed into halfspace range searching in $\mathbb{R}^{d+1}$.

Even for the smallest values of δ, linearization increases the space dimension. Thus, several attempts have been made to deal directly with the original problem on algebraic range shapes. A first question in that direction is concerned with the complexity of arrangements of algebraic surfaces in d-dimensional space. In contrast with the hyperplane case, it is not known whether the complexity of this problem is $O(n^d)$. Chazelle *et al.* [35] proved that this is the case for dimensions 2 and 3 and gave a first non-trivial upper bound for higher dimensions. This was later improved by Koltun [65] who proved $O(n^{2d-4+\epsilon})$ as an upper bound, which is quasi-optimal for $d = 4$. The problem of range emptiness with bounded algebraic range shapes is apparently simpler; however, no algorithms have appeared as improvements over the general range searching case. The only exception that we are aware of is a recent algorithm for balls in $\mathbb{R}^3$ presented by Sharir and Shaul [81].

3.11 Open problems

Here, similar to open problems regarding lower bounds, most of the problems that remain open consist in closing the polylogarithmic gap between algorithms and their corresponding lower bounds. The following list includes the most relevant of these and other problems.

- Can a simplex range searching query be answered in $O(\log n)$ time using $O(n^d)$ space?
- Can a halfspace range reporting query be answered in $O(n^{1-\lfloor 1/d \rfloor} + k)$ using linear space?
- What is the complexity of arrangements of algebraic surfaces for dimensions 4 and higher?
- Are there other data structures simpler than simplicial partitions based on hierarchical cuttings for simplex range searching that offer similar space and query time complexities?

4 Practical data structures

Many of the data structures we have reviewed earlier in this survey do not perform very well in practice even when they provide nice theoretical results. All the data structures we have seen so far do not consider the overhead that may be introduced by accessing secondary memory, some of them involve very large constants, or are very difficult to implement, or disregard how the associated complexities get worse as the dimension increases, even polylogarithmic overheads are sometimes a too-high cost to pay when dealing with availability of resources. In the following, we will see some ways to overcome these problems.

4.1 Secondary memory data structures

When the data base in question is too large to fit in main memory, the data must be stored in secondary memory, and then uploaded to or downloaded from main memory by blocks (of constant size B), as required. Unfortunately, this process is very expensive: a single secondary memory access may take as much time as millions of arithmetic operations. Thus, multiple efforts have been directed to constructing data structures that minimize the number of required memory accesses. This issue has been largely explored for orthogonal range searching. A classic example is the family of R-Tree as originally presented by Guttman [59], and subsequent versions (R^+-tree, R*-tree). What makes an R-tree a suitable data structure to work with secondary memory is the underlying balanced tree with nodes of size B, namely B-trees.

One may think that a good solution to non-orthogonal range searching would be to perform orthogonal range searching as a first step using the d-rectangle r bounding the original query range q and then filtering the output elements to obtain only those that in fact are in q. This can be considered as a heuristic, but not a good general solution since it would require reporting all elements in r as a the first step. Besides, q can be much smaller than r in some cases -as well as the number of elements contained in it. However, the orthogonal case provides a lower bound to simplex range searching. Subramanian and Ramaswamy [83] obtained lower bounds for orthogonal range reporting in 2 dimensions using secondary memory. They proved that in order to answer such queries with only $O(\log_B^{O(1)} n + k/B)$ disk accesses, $\Omega((n/B \log(n/B))/\log\log_B n)$ space must be used, where k is the output size.

Dimension	Query I/Os	Space
2	$O(\log_B n + k)$	$O(n)$
3	$O(\log_B n + k)$	$O(n \log_2 n)$
	$O(n^\epsilon + k)$	$O(n \log_B n)$
d	$O(n^{1-1/\lfloor d/2 \rfloor + \epsilon} + k)$	$O(n \log_B n)$
	$O(n^{1-1/d+\epsilon} + k)$	$O(n)$

Table 3: Upper bounds for halfspace range searching in external memory.

Another way to minimize the number of accesses to secondary memory on non-orthogonal range searching is through the direct application of B-trees as an underlying data structure. Günter [60] considered this approach in the so-called cell-trees. Although the nodes of the cell-tree are partitioned using a generic binary partition scheme[13], they can be constrained to follow the partitioning scheme used on the J-way partition tree of Willard's as presented in section 3.2. In this case, J should be proportional to the block size B instead of the optimum value 3 as we saw before for range searching in main memory (see Figure 1). Agarwal *et al.* [4] provide an optimal solution to halfspace range searching in dimension 2 and also improve previous results for this problem in higher dimensions by adaption of Matoušek's partition trees to secondary memory (see table 3 for a summary of their results). Note an interesting fact: in 2 dimensions, the space complexity of halfspace range searching is smaller than the lower bound provided above for orthogonal range searching, and consequently for simplex range searching, while the query times are similar. This differs from the results on data structures for internal memory when accounting for general operations, instead of number of access to disk.

4.2 Reduction to lower dimensions

We have seen earlier in this survey how the dimension number often appears as an exponent in the complexities associated to range searching. This clearly means that query times and space requirements increase exponentially with the dimension. Thus, a technique that has been explored in order to overcome this issue is to map a higher-dimensional problem to a lower dimensional one. We have already seen a similar idea as used in algorithms based on spanning paths (section 3.4). Spanning paths define a complete 1-dimensional order on the set of input points P, which makes possible their storage as B-trees.

A heuristic approach to range searching related to spanning paths is the use of *space filling curves* (SFC)[14] (see Figure 8 for examples). As P is not necessarily a set of lattice points, we need to divide the space into a grid of buckets containing the points. Then we can apply the SFC of choice on the grid cells. These curves realize fairly good distance-preserving maps, that is, points that are close together in $\mathbb{R}^d$ will be mapped to nearby locations along the 1-dimensional curve. Consequently, similar points are likely to be stored in the same disk block. This application of SFC was introduced by Faloutsos [48]. See Aref *et al.* [9] for an analysis of different types of SPF. Many heuristics based on SFC have appeared since Faloutsos'. A recent example

[13] A binary partition scheme is also used on BSPs (Binary Space Partition). A BSP defines a recursive binary subdivision of the d-dimensional space where each non-leaf region is split by a $d-1$-dimensional hyperplane.

[14] The discrete version of a space filling curve is defined as a 1-dimensional curve that passes exactly once through every single vertex of a lattice set in $[0,1]^d$. Furthermore, these curves usually present regularities such that the curve follows the same behavior at different resolutions.

on *nearest-neighbor*[15] queries has been presented by Chen and Chang [39]. However, a shortcoming to using SFC in general is that the performance of the data structures is subject to the distribution of the points in space.

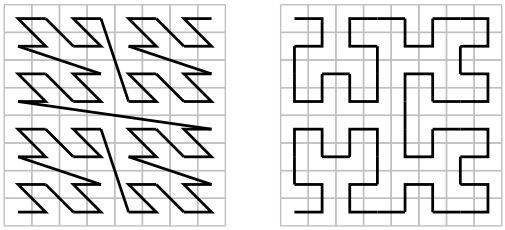

Figure 8: Space filling curves on a planar square grid. Left: Peano (Z). Right: Hilbert.

Other techniques used to reduce the dimension number are based on *hashing*, in particular, locality-preserving hashing functions. Locality preserving hashing functions [62] have been seen as an alternative to 1-dimensional ordering and B-trees. The reason for this is that B-trees may store pointers to objects instead of objects themselves. In our case, this is less of a problem since we are dealing with low-dimensional points and it is preferable to store the points directly on the B-tree nodes at the expense of a small increment of the tree height. Nevertheless, there are hashing-like solutions that address non-orthogonal range searching problems effectively. Several solutions of this type have appeared for the related k-nearest neighbour problem [78, 66, 92], based on the pyramid technique. The pyramid technique, introduced by Berchtold *et al.* [19], consists in partitioning the d-dimensional space $[0,1]^d$ (unit hypercube) into $2d$ hyperpyramids with tops at $(0.5, 0.5, \ldots, 0.5)$ and bases on each of the $2d$ faces of the unit hypercube (see Figure 9). Each point is assigned a hash value equal to the number corresponding to the hyperpyramid it lies in plus the distance from the point to the top of the hyperpyramid. The points are then stored according to their hash values and stored in a B^+-tree for optimal querying.

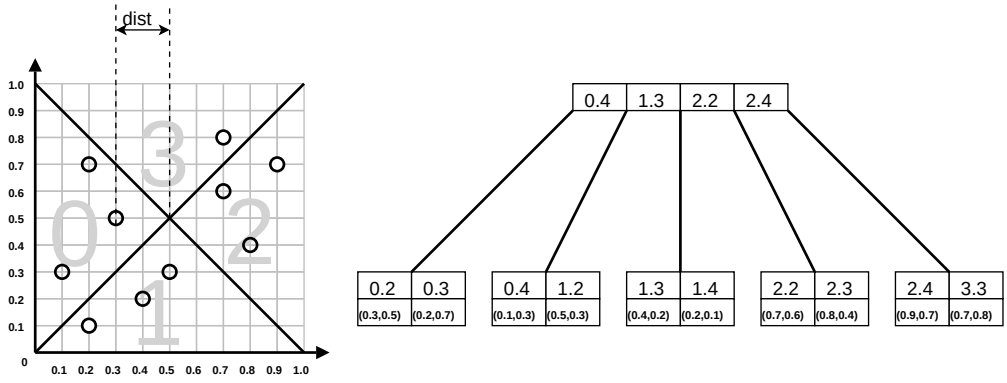

Figure 9: (Adapted from [78]) Pyramid technique. Left: labeled pyramids associated to the 2-dimensional unit square. Right: B^+-tree containing the points (at the leaves).

[15]The problem of nearest-neighbor queries is closely related to ball range searching. For instance, the answer to a nearest-neighbor query trivially leads to the answer to a ball emptiness query.

5 Newer problems on range searching

In this chapter we will present other scenarios where range searching has been (more recently) studied. In some of these scenarios, only results on orthogonal range searching have been published. Therefore, this section entails a new set of open problems on non-orthogonal range searching on its own.

5.1 Parallel and distributed search

One way to approach range searching is parallel computation. In order to parallelize range searching, the dataset is distributed among multiple memory devices (disks) such that multiple processing units (processors) can access data simultaneously. A logical way to distribute the data consists in placing similar points (in terms of coordinates) in different disks. This way a range query targeting a specific area produces a fair load balance among processors. This strategy has been termed as *declustering*. Early studies on declustering for parallel range searching date back to the beginning of the 90's [56, 64, 49]. However, newer declustering techniques have been found and there has been plenty of recent attention to this topic. Ferhatosmanoğlu and other authors [50, 51], for instance, have presented techniques for declustering with small amounts of data replication, and integrated solutions for inter-disk declustering based on hyperpyramidal partitions, together with intra-disk clustering. They have shown empirical evidence of the advantages on using their techniques.

5.2 Moving points

There are scenarios in which the objects (points) are allowed to move. Given a query range r A typical query on moving points can be expressed as: "Report the points that will lie inside of r one minute from now". Early solutions to this problem [82] considered data structures that allowed frequent updates efficiently, so, the motion is simulated by periodic updates. In general, this solution is not the best one can hope for because of the large number of updates required. More recent solutions tend to store functions that describe the movement of the points over time instead of static coordinates.

The trajectory of the points can be approximated by a polygonal path composed of straight line segments on which the points move with constant direction and velocity. At each vertex of the polygonal trajectory of a point, the data structure is updated to adapt to the new movement description. This considerably reduces the number of updates compared to the previous solution. This idea was introduced by Basch *et al.* [15] who provide a framework for the creation of *kinetic data structures*, query types, etc. -although they allow more flexibility towards the description of the motion. Agarwal *et al.* [5] present efficient schemes for a 2-dimensional scenario as described above that allows to answer *recent-past*, *near-future*, and small time interval queries. They consider orthogonal and approximate nearest-neighbour queries.

5.3 Data streams

There are some situations in which the amount of data is too large compared to the available memory - including main, secondary, and all types of memory. It can also be the case that there is data constantly generated, by sensing for instance. In either case, it is assumed that the dataset can only be read once by

the processing unit as a *data stream* but it can not be stored. So, the data needs to be summarized so only a small amount of information is stored for answering subsequent queries. This restriction can be slightly relaxed such that the data can be read a bounded number of times. There are several techniques that can be used for this purpose, for example, extracting the *histogram* of the dataset, computing *aggregate functions*, *randomly sampling* data, or considering only the data within certain *sliding window*. See the survey by Babcock *et al.* [13] for these and other techniques, as well as multiple applications of algorithms on data streams.

Range searching on data streams can not be computed exactly since at least $O(n)$ space would be required for it. A lower bound on the data structure size has been provided by Bagchi *et al.* [14]. They proved that to ϵ-approximate an orthogonal (2-sided) range counting, the space needed is $\Omega(n/\epsilon^2)$, where n is the size of the stream. By ϵ-approximate we mean that an approximate answer x satisfies $x^* \cdot \epsilon \leq x \leq x^*/\epsilon$, where x^* is the exact answer ($0 < \epsilon < 1$). Suri *et al.* [84] give deterministic and randomized ways of sampling the dataset for computing approximate orthogonal and halfspace range counting. They also show how the space required can be reduced by doing more than one pass, in particular, for a number of passes equal to the dimension of the space.

5.4 Wireless sensor networks

A wireless sensor networks (WSN), as the names indicates, consists of a set of sensors (nodes) which are spread over a geographic area and communicate with each other through a wireless network. The objective of a WSN is to keep track of certain events that occur in the sensing area. WSNs provide easy-to-deploy and low cost smart environments that can be applied in multiple fields such as weather analysis, detection of forest fires, studies of animal populations, and all sorts of studies on remote areas, among others. However, the flexibility and versatility of WSNs comes along a set of different problems. The reader is referred to the book by Zhao and Guibas, [93] for a survey on multiple topics related to WSN.

The reason we pay special attention to WSNs is that all the issues previously presented in this chapter, apply to WSN scenarios: in a WSN computations occur in parallel and in a distributed way, objects and sensors may be in motion, and the capacity of storage is very limited for which sometimes the data can be only treated as a stream. Other issues that make range searching on WSN different from more traditional scenarios are failures, costs and delays associated to communication, and measurement errors. Thus, we need to consider some other performance metrics associated with range searching algorithms in this scenario: these include *accuracy*, *resource usage*, and *fault tolerance*. Also, in order to support the dynamics of WSN, the types of queries can be enriched with a new classification according to the four following classes: *historical* queries (queries about the past), *snapshot* queries (queries about the present), *near-future* queries, and *long-running queries* (queries that span an interval of time where a reporting periodicity is sometimes specified). In the following, we look at aspects of range searching data structures in relation with these issues.

As seen earlier in this survey, an ideal data structure for range searching has the topology of a partition tree with a single root node. This can not be directly applied to WSN because of the high load the root node would be subject to: on one hand the root becomes a communication hot spot, on the other hand, if it fails, the entire data structure becomes useless. A naive approach to answering queries on a WSN consists in flooding the whole network starting from the querying point and collecting all the responses coming back from the involved areas of the network. However, this is too expensive in terms of communication resources. There are some approaches that allow range querying more efficiently. One idea is to use *geographic hashing*

which allows setting geographically localizable rendezvous points where event notifications and queries meet. In particular *locality preserving* geographic hashing [63] is very efficient because similar data (events) is stored in nearby sensors.

The volume of data generated may exceed the network storage capability -here we consider storage on the sensors, not on a centralized warehouse. Thus, some sort of aggregation or summarization is needed similarly to dealing with data streams. The aggregation can be done in a hierarchical way. This is the approach considered in DIFS (Distributed Index for Features in Sensor networks) as proposed by Greenstein *et al.* [57]. DIFS consists of a data structure similar to a quadtree but with multiple "roots", and children with multiple parents, in order to avoid hot spots. DIFS has been used as a data structure for answering orthogonal range queries.

A more general approach to introducing higher connectivity and redundancy on a data structure in order to reduce hot spots and provide certain degree of fault tolerance, is *fractional cascading* [55] (see section 3.9 for a definition). Fractional cascading defines a way to interconnect sensors such that every sensor has information about the entire space in a way that fractionally decreases with the distance, but keeping the overall amount of redundancy small ($O(\log n)$ per sensor, where n is the number of sensors).

For halfplane queries in particular, Guibas [58] proposed a heuristic based on a $\sqrt{n} \times \sqrt{n}$ square grid partition of the plane (see Figure 10). Every cell in the partition stores two values corresponding to the partial sums of the values associated to the cells on the left and the cells on the right along its row. The boundary of any query halfplane intersects at most $2\sqrt{n}$ cells. So, a query can be answered in $O(\sqrt{n})$ expected time by a walk along the query line, summing up the precomputed values stored at the cells in each row immediately below (above) the boundary line, together with the weights of points below (above) the line in the crossed cells. This algorithm is the only one known specifically for non-orthogonal range searching in WSN. This suggests the possibility of improvement of non-orthogonal range searching in WSN scenarios. One possible direction to look at is to take advantage of data structures that have originally been design with multiple roots and partitioning schemes, such as those presented by Chazelle *et al.* [36] and Matoušek [73].

6 Conclusions

Non-orthogonal range searching in sublinear time is a hard problem. Memory requirements for query-time-efficient solutions are overwhelmingly large compared to memory requirements of its relative, orthogonal range searching. This has been proved by means of lower bounds on several computational models. However, sophisticated data structures allow a whole spectrum of solutions from sublinear query time with linear space to logarithmic query time with space bounded by a polynomial of degree equal to the dimension of the space. Deterministic and randomized constructions of optimal data structures have been explored. Other data structures have proved to be very efficient in practice, especially under the consideration of secondary memory. Heuristics have also been presented, for example, reducing the problems to a one dimensional equivalent has shown positive results, which has been supported by convincing experimental results. Other problems on range searching have arisen in the areas of wireless sensor networks where several challenges have increased the complexity of more traditional range searching. This field offers a new set of open problems and potential research, especially related to non-orthogonal range searching.

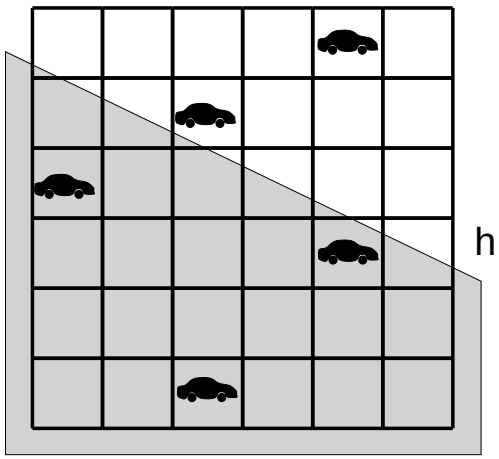

Figure 10: Adapted from [58] Grid used to compute halfspace range searching on WSN and sample query halfplane h.

References

[1] Agarwal, P. K., Partitioning arrangements of lines I: An efficient deterministic algorithm, *Discrete and Comput. Geom.*, 5, pp. 449-483, 1990.

[2] Agarwal, P. K., and Erickson, J., Geometric range searching and its relatives, *Advances in Discrete and Computational Geometry*, in Chazelle, B., Goodman, J. E., and Pollack, R., editors, volume 23 of Contemporary Mathematics, pp. 1-56, AMS Press, 1999.

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

[4] Agarwal, P. K., Arge, L., Erickson, J., Franciosa, P. G., and Vitter, J. S., Efficient searching with linear constraints, *Computer and System Sciences*, 61(2), pp. 192-216, 2000.

[5] Agarwal, P. K., Arge, L., and Erickson, J., Indexing moving points, *Journal of Computer and System Sciences*, 66(1), pp. 207-243, 2003.

[6] Aggarwal, A., Hansen, M., and Leighton, T., Solving query-retrieval problems by compacting Voronoi diagrams, *Proc. of the 22nd Annual ACM Symposium on Comput. Geom.*, pp. 331-340, 1990.

[7] Aho, A. V., Hopcroft, J. E., and Ullman, J. D. *The Design and Analysis of Computer Algorithms*, Addison-Wesley, Reading, MA, 1974.

[8] Alon, N., Haussler, D., and Welzl, E., Partitioning and geometric embedding of range spaces of finite Vapnik-Chervonenkis dimension, *Proc. of the 3rd Annual ACM Symp. on Comput. Geom.*, ACM, New York, pp. 331-340, 1987.

[9] Aref, W. G., Kamel, I., and Mokbel, M. F., Analysis of multi-dimension space filling curves, *Geoinformatica*, 7(3), 179-209, 2003.

[10] Aronov, B., Aurenhammer, F., Hurtado, F., Langerman, S., Rappaport, D., Smorodinsky, S., and Seara, C., Small weak epsilon nets, *Proc. of the 17th Canadian Conf. on Comput. Geom. (CCCG)*, pp. 52-56, 2005.

[11] Arya, S., Malamatos, T., and Mount, D. M., On the importance of idempotence, *Proc. of the 38th Annual ACM Symposium on Theory of Computing*, 2006.

[12] Babazadeh, M., and Zarrabi-Zadeh, H., Small weak epsilon-nets in three dimensions, *Proc. of the 18th Canadian Conf. on Comput. Geom. (CCCG)*, pp. 47-50, 2006.

[13] Babcock, B., Babu, S., Datar, M., Motwani, R., and Widom, J., Models and issues in data stream systems, *Proc. Conf. on Principles of Database Systems*, ACM Press, pp. 116, 2002.

[14] Bagchi, A., Chaudhary, A., Eppstein, D., and Goodrich M. T., Deterministic sampling and range counting in geometric data streams, *Proc. of the 20th ACM Symposium on Computat. Geom.*, pp. 144-151, 2004.

[15] Basch J., Guibas, L. J., and Hershberger, J., Data structures for mobile data, *Journal of Algorithms*, 31(1), pp. 1-28, 1999.

[16] Bentley, J., Multidimensional binary search trees used for associative searching, *Communications of the ACM*, 18(9), pp. 509-517, 1975.

[17] Bentley, J. L., Decomposable searching problems, *Information Processing Letters* 8(5), pp. 244-250, 1978.

[18] Bentley, J. L., Multidimensional divide-and-conquer, *Communications of the ACM*, 23(4), p 214 229, 1980.

[19] Berchtold, S., Böhm, C., and Kriegel, H. P., The pyramid-technique: towards breaking the curse of dimensionality, *Proc. of the 1998 ACM SIGMOD Internat. Conf. on Management of Data*, pp. 142-153, 1998.

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

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

[22] Chazelle, B., Lower bounds on the complexity of polytope range searching, *Journal of the AMS*, 2(4), pp. 637-666, 1989.

[23] Chazelle, B., Lower bounds for orthogonal range searching: I. The reporting case, *Journal of the ACM (JACM)*, 37(2), pp. 200-212, 1990.

[24] Chazelle, B., Lower bounds for orthogonal range searching: II. The arithmetic model case, *Journal of the ACM (JACM)*, 37(3), pp. 439-463, 1990.

[25] Chazelle, B., Cutting hyperplanes for divide-and-conquer, *Discrete and Comput. Geom.*, 9, pp. 145-158, 1993.

[26] Chazelle, B., A Spectral approach to lower bounds with applications to geometric searching, *SIAM J. Comput.*, 27, pp. 545-556, 1998.

[27] Chazelle, B., *The Discrepancy Method*, Cambridge University Press, 2000.

[28] Chazelle, B., and Friedman, J., A deterministic view of random sampling and its use in geometry, *Combinatorica*, 10(3), pp. 229-249, 1990.

[29] Chazelle, B., and Guibas, L. J., Fractional cascading: I. A data structure technique, *Algorithmica*, 1, pp. 133-162, 1986.

[30] Chazelle, B., and Guibas, L. J., Fractional cascading: II. Applications, *Algorithmica*, 1, pp. 163-191, 1986.

[31] Chazelle, B., and Preparata, F. P., Halfspace range searching: an algorithmic application of k-sets, *Discrete and Comput. Geom.*, 1, pp. 83-93, 1986.

[32] Chazelle, B., and Rosenberg, B., The complexity of computing partial sums off-line, *Int. J. Comput. Geometry and Applications*, 1, pp. 33-45, 1991.

[33] Chazelle, B., and Rosenberg, B., Simplex range reporting on a pointer machine, *Computational Geometry: Theory and Applications*, 5, pp. 237-247, 1996.

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

[35] Chazelle, B., Edelsbrunner, H., Guibas, L. J., and Sharir, M., A singly exponential stratification scheme for real semi-algebraic varieties and its applications, *Theoretical Computer Science*, 84(1), pp. 77-105, 1991.

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

[37] Chazelle, B., Guibas, L. J., and Lee, D. T., The power of geometric duality, *BIT Numerical Mathematics*, 25, pp. 76-90, 1985.

[38] Chávez, E., Navarro, G., Baeza-Yates, R., and Marroquín, J. L., Searching in metric spaces, *ACM Computing Surveys*, 33(3), pp. 273-321, 2001.

[39] Chen, H. L., and Chang, Y. I., Neighbor-finding based on space-filling curves, *Information Systems*, 30(3), pp. 205-226, 2005.

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

[41] Clarkson, K. L., Applications of random sampling in computational geometry, II. *Proc. of the 4th Annual ACM Symp. on Comput. Geom.*, ACM, pp. 1-11, 1988.

[42] Cole, R., and Yap, C. K., Geometric retrieval problems, *Information and Control*, 63, pp. 39-57, 1986.

[43] Dobkin, D. P., Hershberger, J., Kirkpatrick, D., and Suri, S., Implicitly searching convolutions and computing depth of collision, *Proc. 1st Annual SIGAL Internat. Symposium on Algorithms*, LNCS 450, pp. 165-180, 1990.

[44] Edelsbrunner, H., *Algorithms in Combinatorial Geometry*. EATCS Monographs on Theoretical Computer Science, 10, Springer-Verlag, Heidelberg, Germany, 1987.

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

[46] Erickson, J., Lower bounds for fundamental geometric problems. *PhD dissertation*, University of California at Berkeley, July, 1996.

[47] Erickson, J., Space-time tradeoffs for emptiness queries, *SIAM J. Comput.*, 29(6), pp. 1968-1996, 2000.

[48] Faloutsos, C., Gray Codes for Partial Match and Range Queries, *IEEE Transactions on Software Engineering*, 14(10), pp. 1381-1393, 1988.

[49] Faloutsos, C., and Bhagwat, P., Declustering using fractals, *Proc. of the 2nd Internat. Conf. on Parallel and Distributed Information Systems*, pp. 18-25, 1993.

[50] Ferhatosmanoglu, H., Tosun, A. Ş., and Ramachandran, A., Replicated declustering of spatial data, *Proc. of the 23rd ACM SIGMOD-SIGACT-SIGART Symp. Principles of Database Systems*, pp. 125-135, 2004.

[51] Ferhatosmanoglu, H., Ramachandran, A., Agrawal, D., and El Abbadi, A., Data space mapping for efficient I/O in large multi-dimensional databases, *Information Systems*, 32(1), pp. 83-103, 2007.

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

[53] Fredman, M. L., A Lower bound on the complexity of orthogonal range queries, *Journal of the ACM (JACM)*, 28(4), 1981.

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

[55] Gao, J., Guibas L. J., Hershberger, J., and Zhang, L., Fractionally cascaded information in a sensor network, *Proc. of the 3rd Internat. Symp. on Information Processing in Sensor Networks*, pp. 311-319, 2004.

[56] Ghandeharizadeh, S., and DeWitt, D. j., Hybrid-range partitioning strategy: a new declustering strategy for multiprocessor databases machines, *Proc. of the 16th Internat. Conf. on Very Large Databases*, pp. 481-492, 1990.

[57] Greenstein, B., Estrin, D., Govindan, Ratnasamy, S., and Shenker, S., DIFS: A distributed index for features in sensor networks, *1st IEEE Ineternational Workshop on Sensor Network Protocols and Applications*, 2003.

[58] Guibas, L. J., Sensing, tracking, and reasoning with relations, *IEEE Signal Processing Magazine*, 19(2), pp. 79-85, 2002.

[59] Guttman, A. R-tree: a dynamic index structure for spatial searching, *Proc. ACM SIGMOD Int. Conf. Management of Data*, Boston, MA, pp. 47-54, 1984.

[60] Günter, O., The design of the cell tree: An object oriented index structure for geometric data bases, *Proc. 5th IEEE Internat. Conf. on Data Engineering*, pp. 589-605, 1989.

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

[62] Hutflesz, A, Six, H. W., and Widmayer, P., Globally order preserving multidimensional linear hashing, *Proc. of the 4th Internat. Conf. on Data Engineering*, pp. 572-579, 1988.

[63] Indyk, P., Motwani, R., Raghavan, P., and Vempala, S., Locality-preserving hashing in multidimensional spaces, *Proc. of the 29th Annual ACM Symp. on Theory of Comput.*, pp. 618-625, 1997.

[64] Kamel, I., and Faloutsos, C., Parallel R-trees, *Proc. of the 1992 ACM SIGMOD Internat. Conf. on Management of Data*, pp. 195-204, 1992.

[65] Koltun, V., Almost tight upper bounds for vertical decompositions in four dimensions, *Proc. of the 42nd IEEE symposium on Foundations of Computer Science (FOCS)*, pp. 56-65, 2001.

[66] Lee, D. H., and Kim, H. J., An efficient technique for nearest-neighbor query processing on the SPY-TEC, *IEEE Trans. on Knowledge and Data Eng.*, 15(6), pp. 1472-1486, 2003.

[67] Lueker, G., A data structure for orthogonal range queries, *19th Ann. Symp. on Foundations of Computer Science (FOCS)*, Ann Arbor, Mich., pp. 28-34, 1978.

[68] Lo, C. Y., Matoušek, J., and Steiger, W. L., Algorithms for ham-sandwich cuts, *Discrete and Comput. Geom.*, 11(4), pp. 433-452, 1994.

[69] Matoušek, J., Construction of ϵ-nets, *Discrete and Comput. Geom.*, 5, pp. 427-448, 1990.

[70] Matoušek, J., Cutting hyperplanes arrangements, *Discrete and Comput. Geom.*, 6(5), pp. 385-406, 1991.

[71] Matoušek, J., Efficient partition trees, *Discrete and Comput. Geom.*, 8, pp. 315-334, 1992.

[72] Matoušek, J., Reporting points in halfspaces, *Computational Geometry: Theory and Applications*, 2(3), pp. 169-186, 1992.

[73] Matoušek, J., Range searching with efficient hierarchical cuttings, *Discrete and Comput. Geom.*, 10(2), pp. 157-182, 1993.

[74] Matoušek, J., Geometric range searching, *ACM Computing Surveys (CSUR)*, 26(4), pp. 422-461, 1994.

[75] Matoušek, J., and Schwarzkopf, O., On ray shooting in convex polytopes, *Discrete and Comput. Geom.*, 10(2), pp. 215-232, 1993.

[76] Matoušek, J., and Wagner, U., New constructions of weak epsilon-nets, *Proc. 19th ACM Sympos. Comput. Geom.*, pp. 129135, 2003.

[77] Moser, W. O. J., Problems on extremal properties of a finite set of points, *Discrete Geometry and Convexity*, Ann. New York Adad. Sci. 440, pp. 52-64, 1985.

[78] Nickerson, B. G., and Shi, Q., K-nearest neighbor search using the pyramid technique, *Proc. of the 18th Canadian Conf. on Comput. Geom. (CCCG)*, pp. 155-158, 2006.

[79] O'Rourke, J., *Computational Geometry in C* (2nd Ed.), Cambridge University Press, 1998.

[80] Preparata, F. P., and Shamos, M. I. *Computational Geometry: An Introduction*, Springer- Verlag, New York, 1985.

[81] Sharir, M., and Shaul, H., Ray shooting amid balls, farthest point from a line, and range emptiness searching, *Proc. of the 16th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pp. 525-534, 2005.

[82] Shekhar, S., and Yang, T., Motion in a geographical database system. *Proc. 2nd Symposium Spatial Database Systems* , LNCS 525, pp. 450-460, 1991.

[83] Subramanian, S., and Ramaswamy, S., The P-range tree: A new data structure for range searching in secondary memory, *Proc. of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 378-387, 1995.

[84] Suri, S., Toth, C. D., and Zhou, Y., Range counting over multidimensional data streams, *Discrete and Comput. Geom.*, 36(4), pp. 633-655, 2006.

[85] Tarjan, R. E., A Class of algorithms which require nonlinear time to maintain disjoint sets, *J. Computer and System Sciences*, 18(1), pp. 110-127, 1979.

[86] Vapnik, V. N., Chervonenkis, A. Ya., On the uniform convergence of relative frequencies of events to their probabilities, *Theory Probab. Appl.*, 16, pp. 264-280, 1971.

[87] Welzl, E., Partition trees for triangle counting and other range searching problems, *Proc. of the 4th Annual ACM Symposium on Computational Geometry*, ACM, New York, pp. 23-33, 1988.

[88] Willard, D. E., Polygon retrieval, *SIAM J. Comput.*, 11, pp. 149-165, 1982.

[89] Yao, A. C., Space-time tradeoff for answering range queries, *Proc. 14th Ann. ACM Syrup. Theory Comput.*, pp. 128-136, 1982.

[90] Yao, A. C., and Yao, F. F., A general approach to d-dimensional geometric queries, *Proc. of the 15th Annual ACM Symposium on the Theory of Communications*, ACM, New York, pp. 163-168, 1985.

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

[92] Zhang, R., Kalnis, P., Ooi, B. C., and Tan, K. L., Generalized multidimensional data mapping and query processing, *ACM Transac. on Database Sys.*, 30(3), pp. 661-697, 2005.

[93] Zhao, F., and Guibas, L., *Wireless Sensor Networks: An Information Processing Approach*, Morgan Kaufmann, (ISBN 1-55860-914-8), 2004.