LOWER BOUNDS IN GEOMETRIC SEARCHING

Burton Rosenberg
(Thesis)

CS-TR-343-91

October 1991

LOWER BOUNDS IN GEOMETRIC SEARCHING

BURTON ROSENBERG

A DISSERTATION
PRESENTED TO THE FACULTY
OF PRINCETON UNIVERSITY
IN CANDIDACY FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

RECOMMENDED FOR ACCEPTANCE
BY THE DEPARTMENT OF
COMPUTER SCIENCE

October 1991

Abstract

We study algorithms for geometric range searching, particularly for the problems of reporting and counting points inside of axis-parallel rectangles and simplices in Euclidean d-space. Lower bounds are discussed as well as the models of computation in which the lower bounds hold. The relevance of these models to practical computing is considered. We then present two new lower bounds for geometric range searching. Related to the problem of computing partial sums off-line, we show that given an array A with n entries in an additive semigroup, and m intervals of the form $I_k = [i, j]$, where $0 < i < j \leq n$, then the computation of $A[i] + \cdots + A[j]$ for all I_k will require $\Omega(\, n + m\,\alpha(m, n)\,)$ semigroup additions. Here, α is the functional inverse of Ackermann's function. Related to the problem of simplex range reporting we prove that given a collection P of n points in d-space, any data structure which can be modeled on a pointer machine and which can report the r points inside of an arbitrary d-simplex in time $O(n^{\delta} + r)$ will require storage $\Omega(\, n^{d(1-\delta)-\varepsilon}\,)$, for any fixed $\varepsilon > 0$. Both of these lower bounds are tight within small functional factors.

Acknowledgments

Before the gentle reader wanders farther into this forest, he should be warned how many woodcutters there are inside. There is the principal author, happy to be removing a few small trees fallen with small and newly trained hands. There is this author's advisor, Bernard Chazelle, who has skillfully marked those trees ready for falling, and landed a heavy blow himself when progress dimmed. There is this author's professor of mathematics, Nicholas Katz, who showed the author how to hone an axe, carry its weight into the cut and gauge the tree's fall. There are the author's two readers, David Dobkin and Robert Tarjan, who carefully inspected every trunk for defects that would slow the progress of the mill downstream. And there is Laszlo Lovász who on several occasions returned the author to a familiar clearing when the darkening woods had become unrecognizable.

I dedicate this thesis to my father and mother, Max and Florence, and above all to my wife, Madeline, for their love, tolerance, humor and support as they watched this story unfold. And to lost faces, unforgotten.

Tell me *how* you are searching, and I will tell you *what* you are searching for.

Ludwig Wittgenstein
PHILOSOPHICAL REMARKS

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Among the most typical activities in information processing is searching [54]. Order, such as having the data sorted, is desirable primarily because it makes searching more efficient. *Geometric search* is the study of the order inherent in geometric spaces and its exploitation for the purpose of search. In this thesis we discuss lower bounds – essentially a quantification of the freedom from disorder geometric spaces have. Although the interest and aim of the study is with respect to computers as we model them today, in fact, the lower bounds have more to do with the intrinsic structure of geometrical objects and their relationships.

The abstract models of computation used in this thesis have been introduced to make the study of lower bounds easier. Generally, they emphasize the inherent complexity of the search space by allowing unlimited capabilities for solving problems other than those related to exploring this space. In reality, computers do not have unlimitied capabilities, and so real computers are restrictions of these ideal conceptualizations. This makes the lower bounds stronger: they remain true within realistic models of computation, and therefore these lower bounds have practical implications.

This thesis concentrates on the following type of geometric search problem. Given a set of n points in E^d, Euclidean d-space, find the points contained in q, where q is some natural subset of E^d. By "natural" we mean that it is some familiarly occurring object in geometry. The subset q is called a *query* or a *range*. Our research concerns the effect of restricting the queries to some family and preprocessing the point set as to render efficient the search for points inside any query taken from the family. The problem derives its name from the class of queries to which we have restricted ourselves. We have, for example,

rectangular range searching, *simplex range searching*, *halfspace range searching* and *spherical range searching* when the query family is the set of all rectangles, simplices, halfspaces and spheres, respectively.

There are two varieties of range searching. In *range reporting* we report all points inside the query while in *range counting* we simply give a count or, more generally, a sum of weights over the points inside the query. This sum may have two distinct algebraic structures: a *group* or a *semigroup*. A semigroup is a group without subtraction. A good example to keep in mind is the semigroup formed by a set of linearly ordered elements with addition being defined as $a + b = \max(a, b)$. It is not possible to define subtraction for this semigroup. This contrasts with the situation where subtraction is simply forgotten about.

The two original results presented in this thesis are:

1. We give a lower bound for the *off-line partial-sum problem*: Given n points $x_1, \ldots, x_n$ arranged left to right on a line, with weights $w(x_i)$ in an additive semigroup and a set of m intervals I_k on the line, compute the sum

$$\sum_{x \in I_k} w(x),$$

for all $k = 1, \ldots, m$. We show that this requires $\Omega(\, n + m\,\alpha(m, n)\,)$ semigroup additions in the arithmetic model of computation. Here, α is the functional inverse of Ackermann's function.

2. We give a lower bound on the space-time tradeoff of *simplex range reporting* on a pointer machine: Given n points in Euclidean d-space, we preprocess the points so as to be able to report those points contained in an arbitrary query simplex in time $O(n^\delta + r)$, where r is the number of points reported and $0 < \delta < 1$. We show that any such algorithm running on a pointer machine will require space $\Omega(\, n^{d(1-\delta)-\varepsilon}\,)$, for any fixed $\varepsilon > 0$.

These results are joint work with Bernard Chazelle. The first result has appeared in journal form in [29].

The naive solution to geometric searching is to check, one by one, which of the n points in the data set lies inside the query. Either report or form the cumulative sum of the points for which the answer is yes. For this solution, call it algorithm $\mathcal{N}$, the three measures of complexity which we will concern ourselves are easily determined. The *preprocessing time*

$P_{\mathcal{N}}(n)$ is $O(1)$: no computation is performed ahead of time in order to aid query processing. The *space* $S_{\mathcal{N}}(n)$ is $O(n)$: we need only store the n points in order to answer any query. The *query time* $Q_{\mathcal{N}}(n)$ is $O(kn)$, where k is the time needed to decide whether a point is inside the query. Since the query objects are familiar geometric objects, rectangles, spheres, etc., k will be a constant.

We are interested in finding data structures which can speed up the processing of queries by preprocessing the point set beforehand. An algorithm $\mathcal{A}$ proceeds in two steps. Presented with point set P containing n points, the algorithm precomputes a data structure of space $S_{\mathcal{A}}(n)$ in time $P_{\mathcal{A}}(n)$. The resulting data structure is denoted $\mathcal{A}_P$. Presented an arbitrary query q from the family of permissible queries Q, the algorithm computes $\mathcal{A}_P(q)$, which, in the reporting variant is $q \cap P$, in time $Q_{\mathcal{A}}(n) = O(\, f(n) + |\, q \cap P \,|\,)$. In the counting variant, $\mathcal{A}_P(q) = \sum_{p \in q \cap P} w(p)$ is computed in time $Q_{\mathcal{A}}(n) = O(\, f(n)\,)$, where w is a weight function from points to a semigroup. The intuitive meaning of these definitions is that the preprocessing is a one-time setup cost for the data structure. Queries will be answered immediately upon presentation. Either the large number of queries over the lifetime of the data structure or the importance of prompt responses will justify the time and space spent in preprocessing.

These definitions hold for the *static* geometric searching problem. One can also discuss the *dynamic* variant. Instead of simply presenting the point set at first and all at once, insertions to and deletions from the point set are possible. The cost of these operations under worst case or amortized measures will be discussed in the text.

1.1 Motivations

Because it combines computers with geometry, computational geometry enjoys wide application. The usefulness of computers is a perhaps surprising but nonetheless obvious fact of modern life. They are the best tool to use on almost any repetitive task or one which requires the handling of large quantities of similar information. They can perform long, tedious, but otherwise straight-forward calculations without fatigue or error. The practical relevance of geometry, insofar as it is the primary mathematical model of the perceptible world, is not doubted. But it, too, can have surprising application areas: consider how geometry is used to study and understand the solutions of simultaneous linear equations.

Geometric range searching comes up in computer graphics, where it is needed for hidden

surface removal or locating the mouse cursor among windows. Hidden surface algorithms require that at each small pixel in the image, the object closest to the viewer be determined, see Sutherland, Sproull and Schumacker [91] for an overview of hidden-surface removal in pratice; the reader is referred to the following papers for a more computational geometry approach: McKenna [66], Fuchs, Kedem and Naylor [48], Paterson and Yao [79] [80], Reif and Sen [83], Bern [13], Overmars and Sharir [76] and Mulmuley [72] [73] In VLSI, range searching is used to check that design rules are being obeyed and in the display and maintenance of the data base of rectangles that make up a design. The design of VLSI uses rectangles to demarcate implantation sites whose interface forms the active components, Mead-Conway [68], and design rules relate the semiconductor physics to a simple set of geometric relationships between these rectangles. Checking the rules provides an example of geometric search, Lauther [59], Baird [7]. For robotics, range searching is an important part of motion planning, and in pattern recognition and learning theory, geometric ideas lead to the need to understand geometric searching. Statistical data analysis can often be recast as a problem of geometric search, Shamos [88], as can several problems in general database theory. In navigation and aviation, several direct examples can be offered: checking the quadrant for high objects or shallow waters, locating the nearest radio station for triangulation.

A lower bound is somewhat of a negative result, particular lower bounds which establish a problem as being of a high order of complexity. However, by closing definitively an unprofitable avenue of attack, they have a positive effect. Attention is directed away from solving the impossible. Often, by changing the definition or the preconditions of the problem, the lower bound can be circumvented. This has the interesting consequence of making possible predictions about the structure of computation devices whose workings are unknown but whose performance is visible. To the engineer it suggests the most gainful happenstance to exploit to insure an economical solution.

The exploration of lower bounds in computational geometry has resulted in some important concepts and theorems in combinatorics. The exchange of intuitions that occurs by thinking about combinatorial problems in terms of geometric structures can result in greater understanding of both. The connection discovered between k-sets and halfspace range searching is an example of such a situation.

1.2 Models of computation

The following three models of computation will be used in this thesis: the *random access machine*, the *pointer machine* and the *arithmetic model*. Other models exist, such as the *cell probe model* of Yao [105] and the *algebraic decision tree* model introduced by Reingold [84], Rabin [82], Dobkin and Lipton [38], Yao [104], Steele and Yao [89] and Ben-Or [8].

The *random access machine*, or RAM, is the most realistic model, it incorporates the capabilities of a typical modern computer. Since it is used for upper bounds, that is, to describe algorithms, it is important not to give it too much power, but within that general guideline, one is free to endow it with whatever functions appear to be reasonable and convenient. A standard definition can be found in Aho, Hopcroft and Ullman [2].

1.2.1 The pointer machine

The *pointer machine* model goes back to Kolmogorov and Uspenskii [55] [56], with some early work also done by Schönhage [87]. Our description is taken from Tarjan [93], where the model was used to prove lower bounds on the time to compute sequences of unions among disjoint sets. Certain variations on the pointer machine are used by Chazelle [18] for the development of algorithms. This requires that the specific powers of the processing unit be delineated, and it is desirable that they be modest. As a model for lower bounds in reporting-type problems, it is used in the research on orthogonal range reporting by Chazelle [20] and on simplex range reporting by Chazelle and the author, Chapter 4.

We describe the model proposed by Tarjan [93]. A pointer machine manipulates two kinds of objects, *data* and *pointers*. The data objects can be taken from a wide class of commonly understood data types: integers, reals, strings, booleans, vectors, and so forth. Pointers are references to *records* which are stored in memory. A record has a fixed format of named *fields*. Each field is either a *data field*, where one datum of any type can be placed, or a *pointer field*, where one pointer can be placed. Besides this memory, the machine has a finite supply of two kinds of registers, *data registers* and *pointer registers*, and a program consisting of a numbered sequence of instructions. The instructions set is shown in Table 1. The last instruction of every program is a **halt**, and each instruction cost one unit of computation.

For the purposes of the lower bound, we modify this model so as to make computation less expensive. Therefore our lower bound holds in this standard model as well as many

instruction	*description*
$r \leftarrow \emptyset$	Place a null pointer in register r.
$r \leftarrow r'$	Copy r' to r. Registers must be of the same type.
$r \leftarrow r'.n$	Copy field n of memory pointed to by register r' into r.
$r.n \leftarrow r'$	Copy r' into field n of record pointed to by r.
$r \leftarrow f(r', r'')$	Apply function f to values in r' and r'' placing result in r.
create r	Create a new record and place a pointer to it in r.
halt	Halt execution.
if $P(r, r')$ **then go to** I	Conditionally branch, where P can be any predicate, I an instruction number.

Table 1: Pointer machine instruction set.

variants of it. In fact, we do not charge nor consider the mechanism by which the pointer machine computes except for charging one unit for each record accessed. However, we do stipulate that a record can be accessed only by possessing its pointer. More formally, the memory is modeled as a directed graph, and the computation follows edges inside this graph, with one unit of computation charged for each edge traversed. The pointer machine can modify this graph and create new nodes in the graph. To report a point p, the machine navigates through memory in order to discover a node in which p has been placed. A more complete discussion of this machine can be found in Chapter 4.

1.2.2 The arithmetic model

The *arithmetic model* is due to Fredman [45] and Yao [107]. In these papers, it was a question of proving lower bounds for orthogonal range queries. The model has also been used by Chazelle [19] to establish lower bounds in simplex range searching. It has been used by Chazelle and Welzl [31] for algorithm design, but this was to prove the optimality of the lower bound, not to describe a practical solution. It is easily argued that the cost of computing in the arithmetic model is not realistic. It does not require an explanation of or a cost for address calculations, for instance. On a RAM, these calculations are part of the algorithm and they must be specified and have their cost accounted for.

This model is used for the counting version of range searching. During the preprocessing phase, the algorithm computes values of a suite of *generators* $\{ g_1, \ldots, g_m \}$. Each generator is a sum,

$$g_i = \sum_{j \in G_i} \alpha_j w(p_j),$$

where $\{p_1, \ldots, p_n\}$ are the points, w is the weight function, α_j is a positive integer, and $G_i \subseteq [1, n]$ is the set of points on which generator g_i depends. The number of generators is the size of the data structure. Each query q will be computed by taking a subset $G(q)$ of $[1, m]$ and forming the sum over g_j for all $j \in G(q)$. For the algorithm to be correct, it must be true that,

$$\sum_{p \in P \cap q} w(p) = \sum_{j \in G(q)} g_j,$$

for all weight assignments to P. The size of $G(q)$ is taken to be the query time of the algorithm. That is, when answering a query we only charge for additions among the generators, we do not charge for the decision as to which generator to use.

Additional stipulations are put on the model so as to make the results reasonably general and to avoid trivialities. In particular, the semigroup must be *faithful.* When we return to this model in Chapter 3 we shall define it more carefully and address its subtleties.

1.3 Preliminaries on data structures

Some of the results in geometric searching are instances of general techniques in data structures. Bentley [10] and Bentley and Saxe [12] showed how data structures for *decomposable problems* can be used as subroutines inside of a larger construction in order to dynamize and add range restrictions to the original solution. We shall use this result in various places of the thesis, and therefore present it here. A more complete study of general techniques for dynamizing data structures is the work of Overmars [77]. Our interest is just to present the two theorems which will be cited later on in this text.

Let the query relation for query q over data set P be signified symbolically by $Q(q, P)$. Perhaps the result is a boolean, perhaps it is a count. We assume it is an element of an additive semigroup. A decomposable problem over P can be broken into smaller problems and the partial results summed.

Definition 1.3.1 *A* decomposable *data structure is one in which the query relation $Q(q, P)$ can be written as,*

$$Q(q, P) = \sum_{i=1}^{k} Q(q, P_i),$$

where $P = P_1 \cup \cdots \cup P_k$ is any disjoint union, and $\sum$ is some commutative, associative operator with unit.

We first explain how to add *range restriction* to any data structure solving a decomposable search problem.

Let the points p of the data set P have a key $\pi(p) \in \mathcal{L}$, where $\mathcal{L}$ is a totally ordered set. Adding range restriction to a query q means that the query can be performed over any subset of P described by $P|_I = \{ p \in P \mid \pi(p) \in I \}$ where I is an interval in $\mathcal{L}$. If the problem is decomposable, we add range restriction by building a balanced binary tree with the set $\{ \pi(p) \mid p \in P \}$ stored in order at the leaves and attaching to each node v in the tree the data structure for range searching in $P|_I$, where I is an interval in $\mathcal{L}$ containing all elements found in the descendant leaves of v. Any interval I in $\mathcal{L}$ can be decomposed into $O(\log n)$ intervals $I_1, \ldots, I_k$ such that for each i there is a node in the tree carrying a data structure for $P|_{I_i}$. Since the problem is decomposable,

$$\mathcal{A}_P(q) = \sum_{i=1}^{k} \mathcal{A}_{P|_{I_i}}(q),$$

where the sum must be correctly interpreted, according to the nature of the problem. Instead of a binary tree, we can use a k-ary tree and we iterate for each of d coordinates of range restriction:

Theorem 1.3.1 (Bentley [10]) *For a decomposable search problem, given an algorithm $\mathcal{A}$ with preprocessing $P_{\mathcal{A}}(n)$, storage $S_{\mathcal{A}}(n)$ and query time $Q_{\mathcal{A}}(n)$, range restriction along d coordinates can be added. Assuming that $P_{\mathcal{A}}$ and $S_{\mathcal{A}}$ grow at least linearly, and $Q_{\mathcal{A}}$ is monotone increasing, and k an integer, the new algorithm $\mathcal{A}'$ has*

- *preprocessing $P_{\mathcal{A}'}(n) = O(\, P_{\mathcal{A}}(n)(\log_k n)^d \,)$,*
- *storage $S_{\mathcal{A}'}(n) = O(\, S_{\mathcal{A}}(n)(\log_k n)^d \,)$, and*
- *query time $Q_{\mathcal{A}'}(n) = (k \log_k n)^d Q_{\mathcal{A}}(n)$.*

See also Theorem 6 of Mehlhorn [70, page 47], and the work of Willard [101], and Lueker and Willard [103].

This construction has the structure of a binary tree and can be modified to accommodate inserts and deletes using another fairly general technique. Given a node v in a binary tree with ν descendents, λ of which are descendents of the left-son of v, the *balance* of v is λ/ν, the fraction of descendents of v which are left-descendents. A *weight balanced tree of balance* α is a binary tree for which all nodes have balance within $[\alpha, 1-\alpha]$. This structure was

first defined by Nievergelt and Reingold [74]. Inserts create new leaves and new nodes, and deletes destroy them, possibly taking a weight balanced tree out of its allowable balance interval. If α is in the interval $(1/4, 1 - 1/\sqrt{2}]$, it can be shown that the tree can recover from the out-of-balance condition by a simple, local restructuring of internal nodes.

In the case of the data structure with range restriction added, there is a tree which accomplishes the range restriction, and its internal nodes have large data structures attached to them. The restructuring of the tree will require the rebuilding of the lower level structures attached to the nodes. However, it is exponentially rare that the nodes near the root will be restructured, consequently, that large attached structures will be rebuilt.

Suppose that when node v becomes out-of-balance, the rebuilding replaces v with a node of balance inside $[(1+\delta)\alpha, 1-(1+\delta)\alpha]$, where δ is some fixed positive rebalancing parameter. By the nature of a weight balanced tree, if v is at level i, it has between $1/(1-\alpha)^i$ and $1/(1-\alpha)^{i+1}$ leaves below it. Assume that v is at level i and is freshly rebuilt, but a insertions and b deletions passed through v and caused it to have balance less than α, but v is again at level i (it may have left that level and returned). With all these assumptions, a calculation we will not reproduce here, see Mehlhorn [69, page 196], shows:

$$a + b > \frac{\alpha\delta}{(1-\alpha)^i}\,.$$

And many of these insertions and deletions occurred while v was at levels $i-1, i$, or $i+1$. If we credit the pair (v, i) each time an insertion or deletion goes through v while it is at level $i-1, i$, or $i+1$, then over a sequence of m insertions or deletions, at most $3m$ credits will in total be given to pairs (w, i), summed over all w in any intermediary tree.

Hence $B(i)$, the total number of rebalancings of nodes at level i, satisfies,

$$B(i) \leq \frac{3m(1-\alpha)^i}{\delta\alpha},$$

and $C(i)$, the total cost of rebalancing at level i, is bounded by

$$C(i) \leq B(i) P_{\mathcal{A}}(\,1/(1-\alpha)^{i+1}\,),$$

where $P_{\mathcal{A}}(n)$ is the preprocessing cost for a structure with n elements. Starting with an empty tree, we can evaluate this sum for a given preprocessing cost function over all $i = 0, \ldots, \log m$. The result we shall need is:

Theorem 1.3.2 (Mehlhorn) *For a decomposable search problem, with preprocessing* $P_{\mathcal{A}}(n) = O(\, n(\log n)^b \,)$, *any fixed* $b \geq 0$, *the algorithm* $\mathcal{A}'$ *of Theorem 1.3.1, can be dynamized to handle a sequence of* m *insertions and deletions in total time* $O(\, m(\log n)^{b+d+1} \,)$. *The preprocessing, space and query asymptotics of the dynamized version are those of* $\mathcal{A}'$.

1.4 Mathematical preliminaries

A linear space over a field K of finite dimension d is the vector space K^d. The unique zero element 0 is called the *origin* of the linear space. Affine space over K of finite dimension d is a set of points for which the linear space of dimension d over the K acts simply transitively, that is, for every two points x, y there is a unique element of the linear space which takes x to y. The linear space is called the space of translations. Having distinguished a point in affine space, the two are related by the natural bijection sending 0 to the distinguished point. The image of linear subspaces in the affine space by way of this bijection are the *flats*. One-dimensional flats are called *lines*. Flats of codimension one are called *hyperplanes*.

Our attention shall generally be restricted to the d-dimensional Euclidean geometry E^d resulting by taking the field to be the reals. Given a point set in E^d, its *affine hull* is the smallest dimensional flat containing all points in the set. For two points p and q, its affine hull is the line of all points $\lambda p + (1 - \lambda)q$ as λ ranges over the reals. The points for which $0 \leq \lambda \leq 1$ are *between* p and q. A set $S \subseteq E^d$ is *convex* if for all x, y in S, all points between x and y are also in S. The *convex hull* of a set of points $CH(P)$, is the smallest convex set containing all the points. Its relative interior is denoted $CH(P)^\circ$. The convex hull of a finite point set is a *polytope*. The boundary of a polytope, $CH(P) \setminus CH(P)^\circ$, contains polytopes of dimensions 0 through $d - 1$, called the *faces* of the polytope. Dimension 0 faces are called *vertices*, dimension 1 faces are called *edges*, and dimension $d - 1$ faces are called *facets*. Points are *affinely independent* if the affine hull of all of the points has greater dimension than the affine hull of any proper subset of the points. A *simplex* is the convex hull of $d + 1$ affinely independent points in d space. A *polyhedral set* is the intersection of a finite number of halfspaces. It includes the notion of polytope, but also non-compact convex objects which contain an infinite ray. The definition of faces, facets, edges and vertices generalize to polyhedral sets. Additional facts about polytopes will be taken from Grünbaum [49].

Projective space over K of dimension d is the space of one-dimensional subspaces of

K^{d+1}. It is affine space with a "hyperplane at infinity" attached. It is different from affine space (all pairs of hyperplanes meet) but can be covered with $d+1$ copies of affine space. Our interest is only to state a duality result useful for transforming geometric problems. Suppose K is the set of reals. The space orthogonal to a one-dimensional subspace of K^{d+1} is a d-flat, and this correspondence is bijective,

$$\{\lambda v \,|\, \lambda \in K\} \mapsto \{w \in K^{d+1} \,|\, \langle w, v\rangle = 0\}, \quad v \in K^{d+1}.$$

Using this correspondence, any nondegenerate bilinear form becomes a pairing of (projective) points and hyperplanes which preserves incidence, see Samuel [85]. The resulting involutory map $\mathcal{D}$ from points to lines and lines to points such that $\mathcal{D}^2 = 1$ is a *duality transformation.*

Any of the standard affine pieces of projective space omits the hyperplane at infinity (a different one each time), hence a pencil of lines is missing from the range of this transformation. The classical inversion with respect to a circle is this duality when the missing pencil is pointed at the center of the circle p, and p is necessarily sent to the line at infinity by $\mathcal{D}$. Another important duality often used to transform geometric problems is inversion with respect to a paraboloid:

$$(p_1, \ldots, p_d) \mapsto \{(x_1, \ldots, x_d) \in E^d \,|\, \textstyle\sum_{i=1}^{d-1} x_i p_i = x_d + p_d\},$$

see Edelsbrunner [39]. The missing pencil is the family of vertical lines hence their common point is at infinity and is sent to the line at infinity, which is also a member of this pencil. Having extracted this pencil, the position of a point p with respect to a line l, whether it is below, on or above, is well defined and reversed by duality. For instance, if p is below l, then $\mathcal{D}(p)$ is above $\mathcal{D}(l)$.

A *complex* is a finite collection of polyhedral sets in E^d which includes all the faces of its members and such that any two members intersect only along a common face. It is more common to consider the relative interior of each polyhedral set as being the member object of the complex, then the complex is a partition of a subset of E^d. The d, $d-1$, 1 and 0 dimensional members of the complex are called the *cells*, *facets*, *edges* and *vertices* of the complex, respectively. An *arrangement* induced by a set H of n hyperplanes in E^d is the complex formed by taking all the connected components of $E^d \setminus (\cup_{h \in H} h)$ and all faces of each component's closure. Again, each face is replaced by its relative interior. This complex, therefore, is a partition of E^d. A complex with all members the (relative interior of) simplices which partition a polytope is a *triangulation* of the polytope.

Chapter 2

Prior work

The complexity of a geometric searching problem is determined by matching a lower bound with an efficient algorithm with query time of equivalent asymptotic order in the same model. Usually, algorithms are stated in terms of realistic models of computation, such as a RAM, while lower bounds use abstract models, such as a pointer machine. Despite this, agreement is quite good between the upper and lower bounds of the problems treated in this thesis, and it gives a clear indication of the problem's computational complexity.

We will review the literature of geometric range searching, especially for *orthogonal range searching*, *simplex range searching* and *half space range searching*. Both upper and lower bounds are described, because we want to establish what the complexity of range searching is. Although the new work in this thesis is for lower bounds, the upper bounds are just as important for a clear picture of computational complexity.

2.1 Orthogonal range searching

In *orthogonal range searching*, the set of queries Q is the collection of axis-parallel rectangles in Euclidean d-space, and P is a finite set of points,

$$\begin{aligned} Q &= \{\, q \subset E^d \mid q = [a_1, b_1] \times \cdots \times [a_d, b_d] \,\}, \\ P &= \{\, p_i \in E^d \mid i = 1, \ldots, n \,\}. \end{aligned}$$

In the reporting variant, we report all points in $P \cap q$ where q is any rectangle in Q. The counting variant gives the size $|\, P \cap q \,|$, or, more generally, the sum of weights $w(p)$ for all $p \in P \cap q$, w being any function from P into a commutative semigroup.

We first discuss two classical solutions, the *k-d-tree* and the *range tree*. The k-d-tree is simple and uses minimal space: only what is needed to store the input. The range tree is a fully dynamic data structure, that is, one which supports inserts to and deletes from the point set, as well as efficient queries. It is possible to improve on these algorithms if semi-dynamic or static structures are all that is required. We survey some improvements in Section 2.1.2 and discuss lower bounds in Section 2.1.3.

2.1.1 The k-d-tree and range tree

The *k-d-tree*, due to Bentley [9], is a k-dimensional binary tree. A one-dimensional binary tree is constructed by recursively halving a set of ordered elements by its median element. One method to define its k-dimensional generalization is to move circularly through the coordinates as the recursion proceeds.

Let π_i be projection onto the i-th coordinate and, for simplicity, assume that $\pi_i(P)$ is injective for every $i = 1, \ldots, d$. The recursive construction of the tree is described by the function $\mathcal{T}(i, R, P)$ which accepts index $i \in [1, d]$, a rectangle R and point set P as input. Let α be the median value of $\pi_i(P)$ (fixed i), P_l be all $p \in P$ such that $\pi_i(p) < \alpha$, P_r be all $p \in P$ such that $\pi_i(p) > \alpha$. Let p be the unique point in P with $\pi_i(p) = \alpha$. Set $i' = i + 1$ if $i \neq d$, else $i' = 1$. The function $\mathcal{T}(i, R, P)$ returns the tree with the set $\{ p, R \}$ stored at the root; its left-son is the tree $\mathcal{T}(i', R \cap \{ x_i \leq \alpha \}, P_l)$; its right-son is the tree $\mathcal{T}(i', R \cap \{ x_i \geq \alpha \}, P_r)$. Calling $\mathcal{T}(1, (-\infty, \infty)^d, P)$ constructs the full k-d-tree for point set P.

Search on query rectangle q is performed recursively. The rectangle R associated with a node stands in one of four relationships with q. It is either contained in R, contains R, is cut by R or avoids R. If it contains R then report the points at all descendent leaves of the node. If it avoids R, do not search the descendents of the node. In either of the other two cases, recursively search the descendents of the node.

Building the tree takes time $\Theta(n \log n)$ and space $\Theta(n)$. The run time analysis was given by Lee and Wong [60] somewhat later. It requires a careful argument to count the worst case number of nodes visited in the tree.

Theorem 2.1.1 (k-d-tree) *The k-d-tree uses space $\Theta(n)$, can be constructed in time $\Theta(n \log n)$ and answers d-dimensional orthogonal range queries in time $O(n^{1-1/d} + | P \cap q |)$ for the reporting problem. For counting (with additional information stored in the internal nodes) the time is in $O(n^{1-1/d})$.*

In the interest of clarity, we have described a greatly enhanced k-d-tree. It is easy to modify the construction so that it requires no storage beyond recording the input. A binary tree of n nodes can be represented in a RAM without pointers in an array with n rows, Tarjan [94]. Let the array have d columns, one for each coordinate. The point p is stored in the row assigned to its node, but we do not store the rectangle R, as it is easily reconstructed during the search phase.

Another data structure for orthogonal range searching is the *range tree*, due to various researchers: Willard [99], [101], Lueker [61], [103], and Bentley [10], [11]. It is another extension of the binary tree, this time giving polylogarithmic search time with a small additional cost in storage.

Orthogonal range searching is a *decomposable* problem: it is possible to report the intersection $q \cap P$ for some query rectangle q and points set P by reporting $(q \cap P') \cup (q \cap P'')$ where $P = P' \cup P''$ is a disjoint union. Likewise, $\sum(q \cap P) = \sum(q \cap P') + \sum(q \cap P'')$. Hence general techniques are available for adding range restriction to the special case of dimension one. That is, using space $O(n)$ and preprocessing $O(n \log n)$ it is elementary to build a data structure solving interval range reporting in time $O(\log n + r)$ for r points reported, and interval range counting in time $O(\log n)$. Using Theorem 1.3.1 and Theorem 1.3.2,

Theorem 2.1.2 (Range tree) *The range tree solves orthogonal range reporting over a set of n points in E^d in time $O(\,(\log n)^d + r\,)$, for r points reported, and space $O(\,n(\log n)^{d-1}\,)$. It requires $O(\,n(\log n)^d\,)$ preprocessing. Insertions and deletions are supported in $O(\,(\log n)^d\,)$ time in the amortized sense.*

A technique explored by Lueker [61] and Willard [99] improves the run time of the range tree by a factor of $\log n$ for reporting mode. The next to last level of the range tree will be a tree organized according to the y coordinate with, at each node, a list of points organized by x coordinate. The collection of lists associated with the children of a node is a partition of the node's associated list, and the order in each sublist is consistent with that in the list. Given a two-dimensional rectangle, it will be broken into $k \leq 2 \log n$ subrectangles along the y direction, and in each of k nodes associated with a subrectangle, a binary search on the x coordinate will be required. Instead of searching each node independently, Willard and Lueker observed that from the position in the parent list, the position in a child list can be recovered by following a pointer prestored in the correct entry of the parent list. Hence, after searching once in the list at the root, stepping from list to list requires only

$O(1)$ time. Therefore the last two levels of search cost $O(\log n)$, rather than $O(\,(\log n)^2\,)$.

However, for counting in an arbitrary semigroup, the range tree as described is optimal for a fully dynamic data structure. Fredman [45] has shown that a sequence of n deletes, inserts and orthogonal range queries will require $\Omega(\,n(\log n)^d\,)$ time in the worst case in the arithmetic model of computation. This result has been extended to groups by Willard [102], but only under some strong restrictions.

2.1.2 Reporting and counting on a pointer machine

The storage required for range counting in either the RAM or pointer machine models can be reduced by the introduction of *functional data structures*,

Theorem 2.1.3 (Chazelle [18]) *Orthogonal range counting on n points in E^d, for $d > 1$, is possible by an algorithm $\mathcal{A}$ of performance,*

- $P_{\mathcal{A}}(n) = O(\,n(\log n)^{d-1}\,)$,
- $S_{\mathcal{A}}(n) = O(\,n(\log n)^{d-2}\,)$,
- $Q_{\mathcal{A}}(n) = O(\,(\log n)^{d-1}\,)$.

This algorithm runs on a RAM or a pointer machine.

Note that the pointer machine referenced by this theorem does not require infinite processing power; any processor which can perform shifts will be adequate. These data structures begin with range trees, but data which is stored is replaced with procedures that can compute it on-demand. Because the time to reconstruct the information is carefully controlled, the view from outside of the data structure is unchanged.

In the case of orthogonal range reporting, a gap between the RAM and the pointer machine models exists, provably. In Chazelle [18], it was shown that range reporting on n points in E^d in a RAM model of computation takes $O(\,(\log n)^{d-1} + r\,)$ time and $O(\,n(\log n)^{d-2+\varepsilon}\,)$ space, for any fixed real $\varepsilon > 0$. However, as is shown in Chazelle [20], if such an algorithm on a pointer machine has query time $O(\,(\log n)^b + r\,)$, for any arbitrarily large constant b, then the space must be in $\Omega(\,n(\log n/\log\log n)^{d-1}\,)$. By a result of Chazelle [15], this is (almost) tight.

On the other hand, if insertions and queries are permitted, Yao [107] has shown for dimension one and Chazelle [21] has extended the result to arbitrary dimension d that

range counting on a RAM will require $\Omega(\, n(\log n/\log\log n)^d\,)$ time to process $O(n)$ insert and query operations in the worst case.

We conclude this section with a review of Chazelle's algorithm for orthogonal range reporting on a pointer machine in polylogarithmic time and $O(\, n(\log n/\log\log n)^{d-1}\,)$ space, showing the lower bound to be almost tight. The algorithm takes advantage of a principle called *filtering search* which means that search time is partially hidden behind the time to report the result. Consider, as a simple illustration of this principle, the *interval stabbing problem* defined as follows: Given n intervals I on a line, preprocess them so that for any point x the set of intervals of I which contain x can be reported in time $O(\log n + r)$ where r is the number of reported intervals. Using filtering search, Chazelle [15] gave a solution to this problem using space in $O(n)$.

The algorithm will use a solution to the *grounded range search* problem as a subroutine: Given n points in the plane, report all points in a range $[a, b] \times (-\infty, y]$, that is, in an improper rectangle whose lower edge is at minus infinity. McCreight [65] introduced a structure called a *priority search tree* which uses space $O(n)$ and makes possible reporting for grounded range search in time $O(\log n + r)$, for r points reported.

These two problems are fused together to solve orthogonal range reporting in the plane. Partition the n points into approximately $\log n$ sets $P_1, \ldots, P_l$ according to their x-coordinate. Connect the points in each P_i into an chain ascending by y. Flattening each P_i by projecting onto y, these chains becomes sets of intervals. Preprocess for interval stabbing the intervals resulting from flattening all these sets, P_1 through P_l. Also, preprocess each P_i twice for grounded range reporting, once where the grounded edge is off to infinity towards the left and again for a grounded edge off to infinity towards the right. Apply this construction recursively inside each P_i.

To answer a query, check first if it falls inside a P_i. If so, recur. Otherwise, cut the query up into at most $\log n$ pieces, no more than two of which are grounded rectangle queries. For the perhaps empty middle piece, do interval stabbing for the bottom edge of the query rectangle. Throw out those intervals stabbed which belong to P_i not fully inside the query rectangle. For the rest, follow the chain of points in each P_i, beginning with the upper point of the stabbed interval, reporting as each point is discovered, until the query interval is exited from above.

The claimed space bound for two dimensions follows from noting that the number of recursive applications of the data structure is $O(\log n/\log\log n)$ and each level takes space

$O(n)$. The time bound is verified by noticing that the descent of the data structure will take at most $O(\log n/\log\log n)$, and the two grounded queries, the stabbing query and the filtering of unneeded intervals will all take $O(\log n)$ each. We extend this result to higher dimensions using the vehicle of Theorem 1.3.1, setting $(\log k)$ to a constant in algorithm $\mathcal{A}$ below, and to $\lceil \varepsilon \log\log n/(d-1)\rceil$ in algorithm $\mathcal{B}$ below, giving the following result:

Theorem 2.1.4 (Chazelle [15]) *Orthogonal range reporting on n points in E^d for $d > 1$ on a pointer machine can be accomplished by algorithm $\mathcal{A}$ with performance,*

- $P_{\mathcal{A}}(n) = O(\, n(\log n)^{d-1}\,)$,
- $S_{\mathcal{A}}(n) = O(\, n(\log n)^{d-1}/\log\log n\,)$,
- $Q_{\mathcal{A}}(n) = O(\,(\log n)^{d-1} + r\,)$ *for r points reported,*

and by algorithm $\mathcal{B}$ with performance,

- $P_{\mathcal{B}}(n) = O(\, n(\log n)^{d-1}\,)$,
- $S_{\mathcal{B}}(n) = O(\, n(\log n/\log\log n)^{d-1}\,)$,
- $Q_{\mathcal{B}}(n) = O(\,(\log n)^{d-1+\varepsilon} + r\,)$ *for r points reported and $\varepsilon > 0$ any fixed real.*

This shows that the lower bound is tight for $b \geq 1$ and $d = 2$, and for $d > 2$ if $b \geq d - 1 + \varepsilon$. The small gap of $(\log n)^{\varepsilon}$ for higher dimensions remains an open problem.

2.1.3 Lower bounds for static range counting

Lower bounds for static orthogonal range counting have utilized the arithmetic model of computation. This results in extremely strong bounds because this model is very general.

Yao [107] looked at a closely related problem called *dominance searching*. Given a set of n points P in E^d with a weight function w, compute

$$\sum_{p \in P,\, p \prec x} w(p)$$

where $p \prec p'$ for two points $p, p' \in E^d$ if and only if every coordinate of p is less than or equal to the corresponding coordinate of p'. This is a special case of orthogonal range counting. The semigroup of values of w must be "sufficiently general", for instance, it should not permit a simulation of subtraction by addition. It is unknown exactly the advantage the

extra structure of a group might bring. Briefly put, we require the semigroup to be *faithful*, see Chapter 3.

Yao investigated this problem for E^2 and showed that for m units of storage the worst case query time is in $\Omega(\log n/\log((m\log n)/m))$. The first bound to hold in higher dimensions is due to Vaidya [95]. He showed that the worst case query time in E^d over n points is $\Omega((n/m)(\log n)^{d-\theta})$, where $\theta = 1$ if $d = 2$ or 3, and $\theta = 2$ for $d > 3$.

Chazelle [21] has given a lower bound which is stronger than both of these. It gives a space-time trade-off for dominance searching in the arithmetic model which is almost optimal. His result is that query time for n points in E^d given m units of storage will require time in $\Omega((\log n/\log(2m/n))^{d-1})$ for the worst case, and with probability approaching 1 if a query is chosen uniformly at random, for a random set of points. Hence the lower bound is true on average as well.

Separately, Yao [106] had shown the one-dimensional, static range counting problem requires time $O(\alpha(m,n))$ given m units of storage, in the worst case, in the arithmetic model of computation, and that this is optimal. Here α is the functional inverse of Ackermann's function. The author and Chazelle [28] [29] have considered a related question: what if the set of queries is known in advance? That is, if instead of precomputing a data structure for the point set P and measuring worst case behavior of a query presented *on-line*, what if the point set P, of n points, and the query set Q, of m intervals, were presented to the algorithm to be solved *off-line*. We present in Chapter 3 a proof that $\Theta(m\alpha(m,n)+n)$ time is necessary and sufficient for this problem.

The upper bound has an easy generalization to d-dimensional arrays: m rectangle queries in a d-dimensional array, size n along each side, can be answered (summed) in time $O(m\alpha(m,n)^d + n)$. It is an open problem to close this gap of $\alpha(m,n)^{d-1}$ for $d > 1$. For this problem, if the semigroup is a group, a $O(1)$ time, linear size solution is possible. On the line, the preprocessing consists of associating with each $p \in P$ the sum of $w(p')$ for all p' to the left of p. A query can be answered by a single subtraction of two prestored sums. Since we are in the arithmetic model, we need not consider the cost of a binary search to locate the endpoints of the interval with respect to the points of P. In higher dimensional arrays, the inclusion-exclusion formula will guide the subtraction.

2.2 Polygon and simplex range searching

The query set of *simplex range searching* is the collection of all simplices in E^d, that is, the convex hull of any $d+1$ affinely independent points,

$$\begin{aligned} Q &= \{\, CH(q_0, \ldots, q_d) \subset E^d \,|\, q_i \in E^d, \text{ affinely independent} \,\} \\ P &= \{\, p_i \in E^d \,|\, i = 1, \ldots, n \,\}. \end{aligned}$$

For reporting, the algorithm returns the set $q \cap P$ for any $q \in Q$. For counting, the algorithm sums $w(p)$ over all $p \in q \cap P$, where w is any function from P into a commutative semigroup.

Searching for points inside a general polytope reduces to simplex range searching. We therefore concentrate on the problem of simplex range searching. Any polytope on n vertices can be triangulated with $O(n^{\lfloor d/2 \rfloor})$ simplices. More exactly, no more than s simplices are required,

$$s = \begin{cases} \frac{n}{n-\nu}\binom{n-\nu}{\nu} & \text{for even } d = 2\nu, \\ 2\binom{n-\nu-1}{\nu} & \text{for odd } d = 2\nu + 1. \end{cases}$$

We sketch the proof. We consider first the case that the polytope P is simplicial. Recall that this means every face of the polytope is a simplex. Pick any vertex v and consider the collection of simplices,

$$\{\, CH(v, P) \,|\, P \text{ are the vertices of a facet not containing } v \,\}.$$

This collection is a triangulation of cardinality bounded by the number of facets. The Upper Bound Conjecture, which was proven to be true by McMullen, see McMullen and Shephard [67], states that the number of facets of a polytope is not greater than that of the *cyclic polytope* which is given by the above formula, Grünbaum [49, page 63]. If P is not simplicial, it can be made so by perturbing the vertices slightly. Each vertex is "pulled" away from the interior of the polytope resulting in new polytope for which the number of k-dimensional faces, $0 < k < d-1$, does not decrease and the number of vertices stays the same, McMullen and Shephard [67, page 116]. A triangulation for the perturbed polytope is a valid triangulation for P as well. That is, carry this triangulation back to P by "pushing" vertices in the reverse order and direction as they were pulled. (But the simplices that become singular, flatten, during the pushing will need to be deleted.)

2.2.1 Partition trees: classical results

The first algorithm for polygon searching in the plane is due to Willard [100]. He proposed a *partition tree* which organized the recursive partitioning of the point set by a pair of lines and two half-lines such that any line cuts at most four of the six regions. Willard's result was a linear space data structure allowing triangle counting in time $O(\, n^{0.774}\,)$ or triangle reporting in time $O(\, n^{0.774}+r\,)$ for r points reported. Edelsbrunner, Kirkpatrick and Maurer [40] used duality to transform halfplane range searching into point location followed by a table look-up for prestored answers to all essentially different triples of halfplane queries. The query time is $O(\log n + r)$ for r points reported, but the space is a gruesome $O(n^7)$.

The idea of partitioning the data set with lines so that queries cannot cut all of the partitions was extended to three dimensions by F. Yao [109], to four dimensions by R. Cole [35], and to arbitrary dimension by F. Yao and A. Yao [108]. In dimension d, Yao and Yao found point set partitions into 2^d pieces such that any hyperplane is guaranteed to miss one of the pieces. This results in a linear space, $O(\, n^{\log_{2^d}(2^d-1)}\,)$ time algorithm for d-simplex range reporting or counting.

Although if the dimension d is less than five such partitions are possible using a set of d hyperplanes, David Avis [5] showed that for dimension five and higher there exist point sets which cannot be partitioned by d hyperplanes into 2^d nearly equal sets. The partitions of Yao and Yao are therefore not induced by d hyperplanes.

Returning to the case of planar search, Edelsbrunner and Welzl [42] improved the exponent for linear-space solutions by inventing the *conjugation tree*. It is a two-dimensional analog of *Fibonacci search*, an idea originated by Kiefer [53], and further developed by Avriel and Wilde [6], Oliver and Wilde [75], and Knuth [54, page 414].

By balancing the multiway divide-and-conquer according to a Fibonacci recurrence equation, the exponent is reduced to $\log(1+\sqrt{5})-1$ which equals 0.695. The conjugation tree is easily described. First we describe a result on cutting point sets with lines in the plane.

Given a point set P in E^2, and a line L, we say the L *halves* P if the two open halfspaces L_l and L_r of L are such that the two sets $P_l = P \cap L_l$ and $P_r = P \cap L_r$ are of equal size. Given an L which halves P, called the *primary line*, it is always possible to find a line L', called the *conjugate line* to L, which simultaneously halves both $P_l = P \cap L_l$ and $P_r = P \cap L_r$. We sketch a proof: The dual of P is an arrangement of lines in E^2, and the dual of L is a point in the arrangement which has the same number of lines above it as below it. Color the lines above the dual of L blue, and those below it red. Looking at the arrangement induced by

the blue lines only, at each x there is a range of y for which the points (x, y) have equal number of lines above as below. Choosing the maximum such y for each x, as x sweeps from negative to positive infinity, these points describe a polygonal line, which we color blue. Likewise, a polygonal red line is constructed from the arrangement of red lines. If, at the extreme left of the blue line, a point q on the blue line also halves the set of red lines, then the dual line of q is a conjugate line. Else, suppose there are too few red lines above q. Then the red polygonal line is ultimately below the blue polygonal line as one travels leftwards. However, the red polygonal line is ultimately above the blue polygonal line as one travels rightwards, because the order in which the lines of the arrangement intersect a vertical ultimately inverts. Hence the polygonal lines intersect. A similar argument applies if too many red lines are above the point q. The point of intersection dualizes to a conjugate line.

At the root of the conjugation tree choose a primary line which halves the point set P into P_l and P_r, and compute a conjugate line. The left-son of the root receives P_l, the right-son receives P_r, and each son is given a copy of the conjugate line. Son i considers the received line to be the primary line l and finds a new conjugate l_i such that l and l_i quarter P_i, for $i \in \{l, r\}$. The procedure then applies itself recursively to each of the sons. A query investigates at most one child and one grand-child of a node, so the query time follows the recurrence $T(n) = T(n/2) + T(n/4) + O(1)$, or $T(n) = O(n^{0.695})$.

Cole and Yap [36] improved the space required for a polylogarithmic search time, their result is an $O(\log n \log\log n)$ time, $O(n^2/\log n)$ space algorithm. Paterson and Yao [78] improved the result to $O(\log n + s)$ for reporting the s points in the interior of a triangle with an algorithm requiring $O(n^2)$ space and preprocessing time.

The approach of Paterson and Yao is an application of Theorem 1.3.1 applied to the subproblem of reporting all lines in a given set L which intersect a query segment $[p, q]$. The set of n points P is sorted by angle around some far away origin O. A binary tree is built over the points ordered by the angles, and to each node v is associated P_v, the subset of P which are in the leaves below v. A node, therefore contains the points inside a *wedge*. Given any cone C with vertex at O, there is a set of at most $2\log n$ of these wedges such that they partition $P \cap C$.

A triangle can be decomposed with respect to the set of wedges into two *quads*. A quad is the area inside a wedge which is between two lines when the intersection of the lines is outside the interior of the wedge. Hence the problem is reduced to $O(\log n)$ instances of

the simpler problem: Find the points of a wedge which are inside a quad. The dual of this problem is: Given lines L, find all lines intersected by the segment $[q, q']$. Specifically, L is the dual of the set of points inside the wedge, and q and q' are the duals of the two lines which cut the wedge: they are the other two sides of the quad.

This latter problem was solved by Chazelle [16]. His algorithm uses time $O(\log n + r)$ and space $O(n)$ for r lines reported. Paterson and Yao incorporate this into their algorithm, but they do not search independently in each of the $O(\log n)$ quads. Instead, a similar coherency as exploited by Willard [99] and Lueker [61] in the range tree (see Section 2.1.1, page 14) is present here to reduce the total search time to $O(\log n + r)$.

2.2.2 Partition trees by ε-nets

Construction of better partitions resulted from the introduction by Haussler and Welzl [52] of ε-nets. These objects are generalizations of a lemma of Clarkson [33] concerning the approximation of geometric arrangements by a random subset of the arrangement.

A *range space* is a pair (X, Q) where X is a set and Q is a class of subsets of X. For any $N \subset X$ let $N \cap Q$ be the set $\{ N \cap q \mid q \in Q \}$. Let

$$\pi(n) = \max_{N \subset X \text{ s.t. } |N|=n} |N \cap Q|$$

be the *primal shatter function* of the space (X, Q). In Sauer [86] and independently in Vapnik and Chervonenkis [96] it has been shown that either $\pi(n) = 2^n$ or there exists an integer d such that for any n element subset N of X with $n > d$ we have $|N \cap Q| < 2^n$; and in this case, $\pi(n) = O(n^d)$. The smallest such d is called the *Vapnik-Chervonenkis dimension* of (X, Q). By convention, we say that the VC-dimension of (X, Q) is infinite when no such d exists. Further results concerning the VC-dimension can be found in Assouad [4].

There are two related notions, that of the *dual shatter function* and the *space of corridors*. Given two points $x, y \in X$ and a range $q \in Q$, we say that q *separates* x and y if it contains exactly one of these points. Given a subset $R \subseteq Q$ of ranges, X is partitioned into *cells*, a cell being a maximal subset of X for which no two points are separated by a range in R. That is, the partition is characterized by the following equivalence relation $=_R$ on X: $x =_R y$ if and only if for all $q \in R$, either both x and y are in q or neither is. Denote by $\Psi(R)$ the number of cells in the partition induced by R. Then,

$$\pi^*(m) = \max_{R \subseteq Q \text{ s.t. } |R|=m} \Psi(R)$$

is the *dual shatter function* of (X, Q). There exists a d such that $\pi^*(m) = O(m^d)$ exactly when (X, Q) has finite VC-dimension, although d and the VC-dimension are not generally equal. For every range space (X, Q) one can define its *space of corridors* (Q, Q°):

$$Q^\circ = \coprod_{x,y \in X} R_{xy},$$

where R_{xy} is the collection of ranges from Q which separate x and y. So Q° is a set of subsets of Q. The space of corridors is a range space and it has finite VC-dimension exactly when (X, Q) does.

Suppose N is a finite set of X. A subset N_ε of N is an *epsilon-net* for N if for any $q \in Q$ with $| q \cap N | > \varepsilon | N |$, then $q \cap N_\varepsilon \neq \emptyset$. An ε-net is a "good" approximation for the finite set with respect to the queries in the range space. Haussler and Welzl [52] showed that if (X, Q) has VC-dimension d, then any finite $N \subset X$ has an ε-net of size $O(\,(d/\varepsilon)\log(d/\varepsilon)\,)$. In fact, a random subset of this size will most probably be an ε-net. This bound was improved by Blumer et al. [14] to $O(\,(d/\varepsilon)\log(1/\varepsilon)\,)$. See Matoušek [62] [63] and Chazelle and Friedman [24] for more on ε-nets and their construction.

Epsilon-nets make possible the construction of efficient partition trees. For example, the following was proposed by Welzl [98]. Suppose (X, Q) is a range space, (Q, Q°) its space of corridors, and that the dual shatter function obeys $O(m^d)$. Then the space of corridors has finite VC-dimension. Let R be a subset of Q of size k. There exists an ε-net R_ε for R of size $O(\,(1/\varepsilon)\log(1/\varepsilon)\,)$. Setting

$$\varepsilon = Cn^{-1/d}\log n,$$

C an appropriate constant, the ε-net has size $O(n^{1/d})$. Adjusting C, we can have $\Psi(R_\varepsilon) < n$. So, given any $P \subseteq X$, as long as $| P | \geq n$, there exist two points $x, y \in P$ in the same cell with respect to R_ε. This means that $R_\varepsilon \cap R_{xy} = \emptyset$. Because R_ε is an ε-net, then:

$$| R \cap R_{xy} | < \varepsilon k = kn^{-1/d}\log n.$$

In summary, for any $P \subseteq X$ of size n and any $R \subseteq Q$ of size k, there exist two points in P separated by no more than $kn^{-1/d}\log n$ ranges of R.

We build a partition tree by iteratively finding such a pair of points and connecting them with an edge. Each time, we discard one of the two points and double the number of ranges in R which separated these points. In detail: we construct R by taking one $q \in Q$ such that $P \cap q = P_i$ for every subset $P_i \subseteq P$ realizable in the range space. Finding two

points $x, y \in P$ which are infrequently separated, we remove x from P and for every $q \in R$ which separated x and y we find another $q' \in Q$ such that $q \cap P = q' \cap P$ and add that to R. This prevents us from choosing two points if the ranges which separate them already separate many other points. In fact, the result is a tree on P such that any range $q \in Q$ separates $O(n^{1-1/d} \log n)$ pairs of points which are edges in the tree. That is, any query can be answered by putting together a sublinear number of (precomputed) pieces. The remaining details include the locating of the edges which are cut by a query in an efficient manner and the precomputing of tree pieces.

We have followed the presentation of Welzl [98], but the introduction of the ideas of VC-dimension and ε-nets appears first in Haussler and Welzl [52]. They proposed a linear space, $O(n^\alpha)$ time algorithm for simplex range queries, or $O(n^\alpha + k)$ time for simplex range reporting with k points reported, where α is any real such that,

$$\alpha > \frac{d(d-1)}{d(d-1)+1}.$$

This result was subsequently improved by Welzl [98] to a linear space, $O(\, n^{1/2}(\log n)^3\,)$ time algorithm for simplex searching in the plane, or $O(\, n^{1/2}(\log n)^3 + k\,)$ time for reporting, where k is the number of points reported. He also gave an $O(\, n^{2/3}(\log n)^3\,)$ time algorithm for three dimensions, but $O(n \log n)$ space was required. These results were extended by Chazelle and Welzl [31] to linear space and $O(\, n^{1-1/d}\alpha(n)\,)$ time in dimension d, but only in the arithmetic model of computation. They also show such a result for spherical range searching. Subsequently, the result was improved by Chazelle, Sharir and Welzl [30] to an $O(\, n^{1+\varepsilon}/m^{1/d}\,)$ time family of algorithms using $O(\, m^{1+\varepsilon}\,)$ preprocessing where m is the amount of space allowed. There is a cognate bound for the reporting variant. This has been surpassed by Matoušek [64] who has given a $O(\, n^{1-1/d}(\log n)^{O(1)}\,)$ time, linear space algorithm requiring $O(n \log n)$ preprocessing.

2.2.3 Random sampling in geometric searching

One instance of Clarkson's lemma [33, Theorem 4.1] involves $\triangle(H)$, a triangulation of E^d *subordinated* to the arrangement induced by H, a set of hyperplanes. By *subordinate*, we mean that the simplices of the triangulation are restricted to having their vertices on vertices of the arrangement and their interiors are not cut by the hyperplanes.

Theorem 2.2.1 (Clarkson [33, Theorem 4.1]) *Given a set H of n hyperplanes in E^d, with probability greater than $1/2$ a random subset H' of H of size r has a subordinate*

triangulation $\triangle(H')$ with the property that every simplex in $\triangle(H')$ has its interior cut by no more than $3d^2 n \log r/r$ hyperplanes from H.

In itself, this fact is enough to establish new algorithms for point location, half-space range searching, Clarkson [33], and simplex range searching, Chazelle, Sharir and Welzl [30].

Our immediate attention is focused on its relation to the notion of ε-nets. We indicate how the random sampling lemma of Clarkson is a special case of this theory. The range space shall be (H, H^*), where H is the set of all hyperplanes in E^d and the queries H^* will be sets hyperplanes. For any simplex S with a distinguished vertex s, the set of hyperplanes separating s from the remaining d vertices of S will be a query in H^*. By duality, this range space is isomorphic with $(E^d, \mathcal{S})$, where $\mathcal{S}$ is the set of all simplices in E^d. Therefore they share the same VC-dimension.

We prove that the VC-dimension of $(E^d, \mathcal{S})$ is $O(d^3)$. The primal shatter function for halfspaces in E^d is bound by n^d: any halfspace which partitions P can be brought into contact with d points of P without changing the partition, thus establishing a surjection from d sized subsets of P onto its partitions. Taking $d+1$ halfspaces, at most $\binom{n^d}{d+1}$ subsets can be created. However, 2^n subsets exist, so for an appropriate c, any $n > cd^3$ implies the existence of a subset of P not realized.

We know, then, that any set of n hyperplanes has a ε-net of size $O(\,(d^3/\varepsilon)\log(1/\varepsilon)\,)$. Setting $\varepsilon = \log r/r$, we conclude that for any set S of n hyperplanes there exists a subset R of size $O(r)$ such that for any simplex, if $(d+1)n\log r/r$ hyperplanes from S intersect the simplex, for some vertex s of S, $n\log r/r$ hyperplanes from S lie in the same neighborhood, therefore some hyperplane from R lies in this same neighborhood. Therefore, the triangulation $\triangle R$ has no simplex cut by more than $O(n\log r/r)$ hyperplanes.

This result can be applied to give a $O(\,(\log n)^{d+1}\,)$ time, $O(\,n^{d+\varepsilon}\,)$ space algorithm for simplex range counting, where $\varepsilon > 0$ is any positive real, Chazelle, Sharir and Welzl [30]. The algorithm exploits the random sampling ideas described in the previous paragraph. Just as an orthogonal query can be decomposed into d intervals, a simplex query can be decomposed into $d+1$ half-spaces, and in both cases a range tree can be used to intersect the component queries. Each half-space query is equivalent to a point location in an arrangement of hyperplanes. Clarkson [33] showed how to do point location by picking a random sample of r hyperplanes, where r is a large enough constant, naively locating the point with respect to the arrangement of the r chosen hyperplanes, then apply this construction recursively in a data structure which processes the $O(n\log r/r)$ hyperplanes crossing the

isolated cell. The geometric decrease in the number of hyperplanes with each level of the recursion assures logarithmic depth to the search tree.

2.2.4 Lower bounds for simplex searching

Fredman [46] [47] gave an $\Omega(\, n^{4/3}\,)$ bound on time for a sequence of n inserts, deletes and queries for any algorithm solving half-plane range queries. His argument has not been extended to higher dimensions. It used the arithmetic model of computation. Chazelle [19] later showed, in this same model of computation but for the static case and for any dimension d, that $\Omega(\,(n/\log n)/m^{1/d}\,)$ time is required, where n is the number of points and m is the amount of storage available. In the plane, a tighter bound was given: $\Omega(\, n/\sqrt{m}\,)$. This can be used to remove the necessity to handle deletes in Fredman's bound, and in general give a time bound of $\Omega(\,(n^2/\log n)^{d/(d+1)}\,)$ for a sequence of n insertions followed by n queries in dimension d, in the worst case. It is seen, then, that the results of the previous section are close to optimal. In the case of linear storage, only factors of $\log n$ remain to be determined, these results are therefore termed *quasi-optimal.* The logarithmic time algorithms are optimal within a space factor of n^{ε}. However, the real moral of the story is that the naive algorithm is the method of choice for these problems.

One contribution of this thesis is to extend the lower bound for logarithmic query time to the pointer model of computation. We show that for a query time of $O(\,(\log n)^b + r\,)$ where r is the number of points reported, a pointer machine will require space $\Omega(\, n^{d-\varepsilon}\,)$ where ε is any positive, fixed real. Previously, no nontrivial bounds for this problem were known.

2.3 Other geometric searching problems

The geometric searching problems mentioned below are not further investigated in this thesis. They are, however, very important problems whose lower bounds are more or less open. In the interest of summarizing the state-of-the-art for lower bounds in geometric searching, we devote a few paragraphs to these results.

2.3.1 Halfspace range searching

Halfspace range searching is intimately connected with point-location and with k-sets. *Point location* is a particularly old problem in computational geometry: Given n hyperplanes in E^d, they form an arrangement of cells, each cell being the set of all points reachable one to the other by a path which does not cross any hyperplane. Having preprocessed the hyperplanes, how quickly can we locate the cell in which lies an arbitrary point $p \in E^d$? Using a duality transform, halfspace range counting is reducible to point location. This was noted by Chazelle, Guibas and Lee [26]; for an enlarged discussion see Stolfi [90] and Preparata [81].

The problem of k-sets is crucial to examining the complexity of halfspace range reporting. It is stated simply: Given a set P of n points in E^d, consider all subsets $P \cap H$ of P with k elements, where H is a halfspace. How many different k-element subsets of P can result from these intersections? Early results on k-sets in the plane can be found in Erdös, Lovász, Simmons and Straus [43]. More on k-sets can be found in Edelsbrunner and Welzl [41], Welzl [97], and Edelsbrunner [39].

Point location was first considered by Dobkin and Lipton [37]. The particular problem of halfspace range search in the plane was addressed by Edelsbrunner, Kirkpatrick and Maurer [40], where range reporting was solved in time $O(\log n + r)$ for r points reported and space $O(n^3)$. Chazelle, Guibas and Lee [26] gave an optimal $O(\log n + r)$ time, $O(n)$ space algorithm. In E^3, Cole and Yap [36] gave an $O(\log n + r)$ time, $O(n^4)$ space algorithm, improved by Chazelle and Preparata [27] to a $O(\, n(\log n)^8(\log\log n)^4\,)$ space, $O(\log n + r)$ time algorithm for reporting r points. This paper established the connection between halfspace range reporting and k-sets and gave new bounds for k-sets in E^3.

In Clarkson [33] [34] the techniques of random sampling and probabilistic methods of Erdös and Spencer [44] were used to give vastly improved results for these two problems. An algorithm for halfspace range reporting over n points in E^d is presented which requires

$O(n^{\lfloor n/2 \rfloor + \varepsilon})$ space, $O(\log n + r)$ time, where r is the number of points reported and ε is any fixed, positive real. It is also established that the maximum value of the sum cardinality of all k-sets with $k = 1, \ldots, j$ for n points in E^d is $\Theta(n^{\lfloor d/2 \rfloor} j^{\lceil d/2 \rceil})$.

The only lower bounds for halfspace range queries are those already mentioned in the previous section: the bound of Fredman [46] [47] and its extension by Chazelle [19].

2.3.2 Spherical range searching and assorted others

Spherical range searching in dimension d can be reduced to halfplane range searching in dimension $d + 1$ by the coordinate transformation,

$$(x_1, \ldots, x_d) \mapsto (x_1, \ldots, x_d, \sum x_i^2).$$

This transformation lifts up E^d onto a surface in E^{d+1} such that the interior of balls in E^d become patches of surface below a hyperplane in E^{d+1}. But it is possible to construct partition trees of spheres. Since the VC-dimension of the dual range space is d, this gives a more efficient algorithm when space is constrained to be linear. Results along these lines are found in Chazelle and Welzl [31].

It is worthwhile to point out some range searching problems which closely resemble those already mentioned but have dissimilar lower bounds. Although orthogonal range queries in E^d require $O(n \log n / \log \log n)$ space for logarithmic query time, if one side of the range lies on a fixed line, then the space is $O(n)$, using the priority queues of McCreight [65]. If the aspect ratio of the rectangle is fixed, Chazelle and Edelsbrunner [23], or if the query is a trapezoid with one side against a fixed line, Chazelle and Guibas [25], then the space will be linear for logarithmic time queries. Also, translates of a fixed convex body in the plane can be range reported in $O(\log n + r)$ time and linear space. Range reporting for a fixed radius disk, Chazelle and Edelsbrunner [22], is a special case of this proposition.

Chapter 3

The complexity of computing partial sums off-line

The *one-dimensional off-line partial-sum problem* is specified by a set X of n variables $x_0, x_1, \ldots\ldots, x_{n-1}$, a weight function w from X to an additive commutative semigroup S, and a set Q of m intervals, $Q \subseteq \{\,[x_i, x_j] \mid 0 \leq i < j < n\,\}$. We must form, for each interval q in Q, the sum,

$$\sum_{x \in q} w(x).$$

The set Q is called the *task* and its elements are called *queries*.

Intuitively, a fast solution to the problem might begin by computing and placing in memory useful partial sums, which can be shared by many queries. Then each query is quickly answered by combining a small number of these partial sums. The lower bound we prove implies that no allocation of partial sums can give solutions using only a constant amortized number of arithmetic operations per query. Note that, for a group, a solution using a constant number of operations per query can be achieved by precomputing all prefix sums and, for any query, subtracting the appropriate two among them.

Partial-sums are a special case of classical orthogonal range searching. We have further distinguished two flavors of partial-sums. In *query mode*, preprocessing is allowed and q is a query to be answered on-line. In *off-line mode*, we are given the set $\{x_1, \ldots, x_n\}$ and a set of intervals $q_1, \ldots, q_m$, and we must compute the m sums $\sum_{j \in q_i} w(x_j)$, for each i. The query mode situation was studied by Yao [106]. He proved that if only m units of storage are permitted then there exists a query requiring time $O(\alpha(m, n))$ to answer. The

function $\alpha(m,n)$ is the functional inverse of Ackermann's function defined by Tarjan [92]. The function α, seemingly esoteric, arises in a number of computer science contexts. Tarjan [92] showed it was related to the complexity of union-find, and it bounds the length of Davenport-Schinzel sequences, Hart and Sharir [51], Agarwal, Sharir and Shor [1]. But our proof does not involve a reduction from these problems.

Our main result is a nonlinear lower bound for the one-dimensional off-line partial-sums problem. We show that there exists m partial sums over the n variables which require $\Omega(n + m\alpha(m,n))$ semigroup additions for evaluation. It is a generalization of Yao's result to the situation where the queries are known ahead of time. It can also be regarded as a generalization of a result of Tarjan [92] concerning the off-line evaluation of functions defined over the paths of a tree. See also Alon and Schieber [3] for related upper and lower bounds.

A matching upper bound has been given by Yao [106] which works in the off-line case as well. So the results presented are optimal. In Chazelle and Rosenberg [28] we extend the upper bound to the case of multi-dimensional arrays: Given a d-dimensional array A and m rectangles, we compute the sum over every rectangle in time $O(n + m\alpha(m,n)^d)$, but we do not know if this is optimal.

The rest of the chapter is organized as follows: in Section 3.1 we define the model of computation and discuss some useful reductions. Section 3.2 is devoted to the construction of hard problem instances, while Section 3.3 gives the proof of the lower bound. We discuss applications and open problems in Chapter 5. A preliminary version of this chapter has appeared in Chazelle and Rosenberg [28] and a journal version is published in Chazelle and Rosenberg [29].

3.1 The arithmetic model of computation

The *arithmetic model of computation*, Fredman [45], Yao [106] [107], Chazelle [21], charges one unit of computation for every semigroup operation performed. All other computation is free. We can store results in memory cells, access the cells by address, using whatever address arithmetic we desire. In other words, a solution is a straight-line program with instructions of the form $z_j = w(x_i)$ or $z_j = az_k + bz_l$, a and b integral, where $z_0, z_1, \ldots$ form an unbounded set of variables. At the end of the computation we require that the m partial sums specified by the task should be given by $z_0, \ldots, z_{m-1}$.

In the arithmetic model of computation the cost of a solution is simply the number of instructions of the form $z_j = az_k + bz_l$. This cost is referred to as the *time* for the solution. To make our lower bounds more general, we require that a solution should work regardless of the particular assignment of weights to the x_i's. Also, we must assume that the semigroup is not trivial, unlike, say, the semigroup $(\{0\},+)$. Following Yao [107] we define a semigroup $(S,+)$ to be *faithful* if the identity of two linear forms implies the equality of their sets of variables. That is to say, given two sets of indices I and J, and non-zero integers a_i, b_j, the relation,

$$\sum_{i\in I} a_i y_i = \sum_{j\in J} b_j y_j$$

cannot be an identity unless $I = J$. Note that we do not even require that the sets $\{a_i\}$ and $\{b_j\}$ should be the same. Examples of faithful semigroups are $(\mathbf{Z}, \max)$, $(\{0,1\},\wedge)$, $(\{0,1\},\vee)$ and $(\mathbf{Z},+)$. But note that $(\{0\},+)$ and $\mathbf{Z}/2\mathbf{Z}$ are not faithful.

Given a set X, its power set is denoted $\mathcal{P}(X)$ — it is the set of all subsets of X. The algebraic structure $(\mathcal{P}(X),\cup)$ is a commutative semigroup. It has the additional properties of being idempotent, $a \cup a = a$ for all a in $\mathcal{P}(X)$, and possessing an identity element, $a \cup \emptyset = a$ for all a in $\mathcal{P}(X)$. As long as X is not itself empty, $(\mathcal{P}(X),\cup)$ is faithful. We are interested in this semigroup for the following reason. Any solution to a task Q with weight function w into a faithful semigroup can be interpreted as a solution to the same task Q with weight function into the semigroup $(\mathcal{P}(X),\cup)$ defined by,

$$\begin{array}{rccc} w^*: & X & \rightarrow & (\mathcal{P}(X),\cup) \\ & x & \mapsto & \{x\} \end{array}$$

simply by replacing w in the solution with w^*. We make the following definition:

Definition 3.1.1 *A scheme S is a sequence $s_0,\ldots,s_{r-1}$ of subsets of X such that for all $i \in [0,r-1]$, $s_i = s \cup s'$, where $s = s_j$ for some $j < i$ or $s = \{x\}$ for some $x \in X$ or $s = \emptyset$, and likewise for s'.*

We say that a scheme S solves task T if all partial sums in T with weight function w^* occur as elements in S.

Lemma 3.1.1 *Let T be a task over n variables and S a scheme of minimum length solving it. Then, for any faithful semigroup, a solution to T with weight in the semigroup takes time at least $r - n$, where r is the length of the scheme.*

Proof: Recall that in the arithmetic model of computation a solution to T is a straight-line program. Each line of the program gives rise to a subset of X by replacing w with w^*, and $+$ by $\cup$; hence, the entire program gives rise to a sequence of subsets $s_0, s_1, \ldots$, which obviously satisfy the definition of a scheme. Since the program is a solution to T, for every $t \in T$ there is an i such that the i-th line in the program computes the sum,

$$\sum_{x \in t} w(x).$$

By faithfulness, and the fact that the solution works regardless of the weight assignment, s_i is $\{ x \in t \}$. Since this set is $w^*(t)$, we conclude that S solves T. The length of the scheme is the number of lines in the program. We need no more than n of these to be of the form $z_j = w(x_i)$. The result follows. □

We give a few more definitions before beginning the real work. Let $[i, j]$ denote the set of all integers between i and j inclusive. Let X be a finite set whose n elements are denoted $x_0, x_1, \ldots, x_{n-1}$. Any subset of X of the form $\{ x_k \mid k \in [i, j] \}$ is called an *interval* and is (abusively) denoted $[x_i, x_j]$. Intervals of the form $[x_i, x_i]$ are called *trivial.* The empty set, by convention, is also a trivial interval.

In the proofs that follow, we will define mappings between schemes. These are often simply maps between sets $f : X \to Y$ extended to maps between powersets $f : \mathcal{P}(X) \to \mathcal{P}(Y)$ in the usual way: requiring that $f(A \cup B) = f(A) \cup f(B)$. Other times, we intend that the map be between intervals. We denote by $\mathcal{I}(X)$ the set of all intervals in $\mathcal{P}(X)$,

$$\mathcal{I}(X) = \{ [x_i, x_j] \subseteq X \mid i \leq j \}.$$

A map $f : X \to Y$ extends to $f : \mathcal{I}(X) \to \mathcal{I}(Y)$ by $f([x_i, x_j]) = [f(x_i), f(x_j)]$. It is often convenient to define a map by defining its inverse first. A *section* of a map $f : X \to Y$ is a map $g : Y \to X$ such that the composition $f \circ g$ is the identity on T. For example, a map from $X = \{ x_0, \ldots, x_{kn-1} \}$ to $Y = \{ y_0, \ldots, y_{n-1} \}$ which takes x_i to $y_{i \bmod n}$ has k sections which look like $y_j \mapsto x_{in+j}$ with $i \in [0, k-1]$, fixing a different i for each section.

3.2 Constructing hard tasks

We will construct a family $\mathcal{T}_n(t, k)$ of hard tasks parametrized by two integers $t \geq 1$ and $k \geq 0$, respectively called *time* and *density*; the subscript n indicates the number of variables and is not a parameter. Task $\mathcal{T}_n(t, k)$, defined over $\{ x_0, \ldots, x_{n-1} \}$, will contain between

$kn/2$ and kn queries, and any scheme solving it must be of length at least $tkn/6$. That such a family exists at all puts a lower bound on the cost of computing partial sums. Define the function $R(t,k)$, for all integers $t \geq 1$ and $k \geq 0$:

$$\begin{aligned} R(1,k) &= 2k, && k \geq 0, \\ R(t,0) &= 3, && t > 1, \\ R(t,k) &= R(t,k-1)R(t-1,R(t,k-1)), && k > 0,\ t > 1. \end{aligned}$$

This function gives the n needed to construct the hard task $\mathcal{T}_n(t,k)$.

Lemma 3.2.1 *For all integers $t \geq 1$ and $k \geq 0$, there is a task $\mathcal{T}_n(t,k)$ over the n element set $X = \{ x_0, \ldots, x_{n-1} \}$ satisfying the three requirements:*

1. $|\mathcal{T}_n(t,k)| \geq kn/2$, *where* $n = R(t,k)$,
2. $|\{ [x_i, x_j] \in \mathcal{T}_n(t,k) \mid i = l \}| \leq k$ *for any* $l \in [0, n-1]$.
3. *If* $S = \{ s_0, s_1, \ldots, s_{r-1} \}$ *is a scheme solving* $\mathcal{T}_n(t,k)$, *then* $r \geq t|\mathcal{T}_n(t,k)|/3$.

All intervals in $\mathcal{T}_n(t,k)$ are nontrivial.

The second requirement is called the *uniform right-degree condition.* Aside from implying that $\mathcal{T}_n(t,k)$ contains no more than kn intervals, it is an induction invariant crucial to the inner workings of the construction.

The proof of this lemma is split over the remainder of this section and the next. In this section the construction is delineated and the first two requirements verified. The next section deals with the last requirement. The lemma easily leads to the lower bound theorem stated and proved in the final half of the next section.

The construction is by double induction on t and k. We present directly tasks for $t = 1$ and $k \geq 0$ and for $k = 0$ and $t \geq 1$. This is the basis of the induction. The task $\mathcal{T}_n(t,k)$ is constructed from $\mathcal{T}_{n'}(t,k-1)$ and $\mathcal{T}_{n''}(t-1,n')$, for the inductive step. We sketch the idea behind the double induction.

Suppose we have a hard task and that in this task it often occurs that many intervals have their left endpoint over the same variable. That is, one large group of intervals ends over x_1, another over x_2, and so on. If we replace each x_i by a group of variables $x_{i,1}, \ldots, x_{i,m}$ and replace each interval $[x_i, x_j]$ by $[x_{i,k}, x_{j,1}]$ where k chosen so that in the left endpoints

are evenly spread out over all the new variables, it is intuitively plausible that the resulting task will require more time to solve than did the original task — because it is a stretched out version of the original task plus new diversity in the left endpoints of intervals. However, the resulting task will have a density m times smaller than the original. This can be remedied without making the task easier by placing into each group of variables $x_{i,1}, \ldots, x_{i,m}$ a task of sufficient density and hardness. We will follow through on this idea in the remainder of the chapter, repeating the steps precisely and proving correct the intuition about the hardness of the resulting task.

Let $\mathcal{T}_3(t,0) = \emptyset$ for all $t \geq 2$. Since $k = 0$, $|\mathcal{T}_3(t,0)|$ is large enough; since $|\mathcal{T}_3(t,0)| = 0$, any scheme solving $\mathcal{T}_3(t,0)$ is long enough; and $\mathcal{T}_3(t,0)$ satisfies the uniform right-degree condition for $k = 0$. Now we define $\mathcal{T}_n(1,k)$ with $k \geq 0$. Let $n = 2k$ and over variable set $\{ x_0, \ldots, x_{n-1} \}$ define:

$$\mathcal{T}_n(1,k) = \bigcup_{i=0}^{n-k-1} \bigcup_{j=1}^{k} [x_i, x_{i+j}].$$

It is easily verified that $\mathcal{T}_n(1,k)$ satisfies the uniform right-degree condition and has size $|\mathcal{T}_n(1,k)| = kn/2$. Since all the queries in $\mathcal{T}_n(1,k)$ must appear in any scheme solving it, the size of $\mathcal{T}_n(1,k)$ is a lower bound for the scheme's length. Hence $\mathcal{T}_n(1,k)$ satisfies the three requirements of the lemma.

We now assume that $k > 0$ and $t > 1$. By induction hypothesis, we have tasks $A = \mathcal{T}_a(t,k-1)$ and $B = \mathcal{T}_b(t-1,a)$ where $a = R(t,k-1)$ and $b = R(t-1,a)$. Since $R(t,k) = ab$, we intend to construct task $Q = \mathcal{T}_n(t,k)$ over $n = ab$ variables. For clarity, we give different names to all these different variable sets. Name the variables in Q by $X = \{ x_0, \ldots, x_{n-1} \}$, those in A by $Y = \{ y_0, \ldots, y_{a-1} \}$, those in B by $Z = \{ z_0, \ldots, z_{b-1} \}$.

Divide X into b blocks each containing a consecutive variables. Into each block, we place a copy of the task A. To state this formally, we define the map:

$$\begin{array}{rccc} \varphi: & \mathcal{P}(X) & \to & \mathcal{P}(Y) \\ & x_i & \mapsto & y_{i \bmod a} \end{array}$$

and take these sections:

$$\begin{array}{rccc} \varphi_j: & \mathcal{I}(Y) & \to & \mathcal{I}(X) \\ & y_i & \mapsto & x_{ja+i}, \end{array}$$

where $j = 0, \ldots, b-1$. Each section gives a copy of A placed in X, it is the image of A by φ_j. Though φ is a map between subsets of sets, it is defined as if it were a map of sets. Likewise, φ_j is claimed to be a map of intervals, even though we have only given its values

on elements. It is good to reflect on these distinctions, but burdensome to reflect these distinctions in the notation. In any case, $\varphi \circ \varphi_j$ is the identity on Y, as it should be for a section.

To guide what remains to be done for our construction, we mark some of the x_i. Let the leftmost variable in each block be marked, that is, x_{ia} for $i = 0, 1, \ldots, b-1$. Now alter the marking by removing the mark on x_0 and placing it on x_{n-1}.

A copy of task B is placed over variables X guided by these marked variables: variable z_i goes to the i-th marked variable in X. The complete map follows from two desires: that the map be a map of intervals, that no two intervals from B have left ends over the same variable in X. This second requirement means that we have to stretch leftwards by different amounts intervals coming from B. This stretching step is very important since it causes the composite task to be more difficult to solve than the sum difficulty of its components.

If we were to take each interval $[y_i, y_j]$ in B to an interval $[x_{i'}, x_{j'}]$ in Q by the rule "$x_{i'}$ is the i-th marked variable in Q," at most a intervals from B would have their left ends over a given marked x in Q. Further stretch each interval leftward, by a different amount, so that they end over x_{i-1}, x_{i-2}, etc. We formalize this construction as follows. Partition B into a subsets $B_0, \ldots, B_{a-1}$ so that the partition obeys the following restrictions:

1. $|\{\, [z_j, z_k] \in B_i \mid j = c \,\}| \leq 1$ for all $i \in [0, a-1]$ and $c \in [0, b-1]$.
2. $[z_{b-2}, z_{b-1}] \notin B_0$.

It is possible to construct this partition thanks to the uniform right-degree condition on B and the fact that there is only one nontrivial interval ending over z_{b-2}.

For $i \in [0, a-1]$ define the map,

$$\begin{array}{rccl} \psi_i : & \mathcal{I}(Z) & \to & \mathcal{I}(X) \\ & [z_j] & \mapsto & \begin{cases} [x_{(j+1)a-i}, x_{(j+1)a}] & \text{for } j \in [0, b-2] \\ [x_{ab-1}] & \text{if } j = b-1. \end{cases} \end{array}$$

We must define a map of intervals. To this end we shall ensure that $\psi_i([z_j, z_k])$ is the smallest interval containing all of $\{\, \psi_i(z_j), \psi_i(z_k) \,\}$. Each of these is a section of the map,

$$\begin{array}{rccl} \psi : & \mathcal{P}(X) & \to & \mathcal{P}(Z) \\ & x_i & \mapsto & \begin{cases} z_j & \text{if } i = a(j+1), \\ z_{b-1} & \text{if } i = ab-1, \\ \emptyset & \text{otherwise.} \end{cases} \end{array}$$

An image of each B_i is placed over X using ψ_i. Remark that any such interval over X spans blocks. For this to be true, the restriction that $[z_{b-2}, z_{b-1}]$ not be in B_0 is crucial.

The task Q is defined as,

$$Q = \left(\bigcup_{j=0}^{b-1} \varphi_j(A) \right) \bigcup \left(\bigcup_{i=0}^{a-1} \psi_i(B_i) \right).$$

We investigate the properties of this task.

By construction, Q is a collection of nontrivial intervals in X. In fact, each map φ_j and ψ_i is one-to-one. The distinct character of each of these maps assures that each component that went into making Q is disjoint with every other. This implies that

$$\begin{aligned} |\,Q\,| &= \sum_{j\in[0,b-1]} |\,\varphi_j(A)\,| + \sum_{i\in[0,a-1]} |\,\psi_i(B_i)\,| \\ &= b\,|\,A\,| + |\,B\,| \geq b(k-1)a/2 + ab/2 = kab/2, \end{aligned}$$

and

$$\begin{aligned} |\,\{\,[x_i, x_j] \in Q \,|\, i = c\,\}\,| &= |\,\{\,[y_i, y_j] \in A \,|\, i = c \bmod a\,\}\,| \\ &\quad + \left|\,\{\,[z_i, z_j] \in B_{(-c) \bmod a} \,|\, i = \lfloor (c-1)/a \rfloor\,\}\,\right| \\ &\leq (k-1) + 1 = k. \end{aligned}$$

for all $c \in [0, ab-1]$. Therefore Q is a task of the correct density and obeys the uniform right-degree condition.

3.3 The lower bound

We now derive a lower bound on the cost of any scheme S which solves Q. Each of the s_j in scheme S falls into one of $b+1$ categories. Either, for some $i \in [0, b-1]$, s_j lies fully inside block i, $s_j \subseteq [x_{ia}, x_{(i+1)a-1}]$, or s_j combines elements from different blocks. We partition S into $b+1$ sublists according to this categorization. If s_j lies inside block i, place s_j in S^i, else place s_j in S^b. If s_j is the empty set, place it in the subsequence S^b. Maintain the original ordering, but renumber, to obtain sublists $\{\, s_0^i, s_1^i, \ldots, s_{r_i-1}^i \,\}$ where $i = 0, \ldots, b$, each s_j^i is an element of $\mathcal{P}(X)$, and the r_i are the length of these lists, noting $r = r_0 + \cdots + r_b$. It will be shown that each of the subsequences S^i for $i = 0, \ldots, b-1$ is, essentially, a solution to A. Immediately, we have lower bounds for all r_i with i in this range. The subsequence

S^b is essentially a solution to B, but it is a very inefficient solution. We can quantify this inefficiency and, in doing so, derive a lower bound on r_b.

Lemma 3.3.1 *For $i = 0, \ldots, b-1$, the sequences $\varphi(S^i)$, defined by $\varphi(S^i)_j = \varphi(s^i_j)$, are schemes solving A.*

Proof: Let s_α be any member of $\varphi(S^i)$. Then s_α is the image of some element s_j in S. Perhaps $s_j = s_k \cup s_l$ with $k, l < j$. Clearly, s_k and s_l fall into S^i with images s_β and s_γ in $\varphi(S^i)$, where $\beta, \gamma < \alpha$. But

$$s_\alpha = \varphi(s_j) = \varphi(s_k \cup s_l) = \varphi(s_k) \cup \varphi(s_l) = s_\beta \cup s_\gamma.$$

The other possible precursors of s_j are argued similarly, proving that $\varphi(S^i)$ is a scheme. We know that φ_i is a section of φ and that S solves Q. This implies $S^i \supseteq \varphi_i(\mathcal{T}_a(t, k-1))$, giving

$$\varphi(S^i) \supseteq \varphi \circ \varphi_i(\mathcal{T}_a(t, k-1)) = \mathcal{T}_a(t, k-1).$$

So $\varphi(S^i)$ solves $\mathcal{T}_a(t, k-1)$. □

Lemma 3.3.2 *The sequence $\psi(S^b)$, defined by*

1. $\psi(S^b)_{i+1} = \psi(s^b_i)$,
2. $\psi(S^b)_0 = [z_{b-2}, z_{b-1}]$,

is a scheme solving B. It is not minimal: there is a subsequence of $\psi(S^b)$, resulting from the removal of $|B| - |B_0|$ elements from $\psi(S^b)$, which is also a scheme solving B.

Proof: Consider any s_α in $\psi(S^b)$. It is an element of S, say s_j. Assume that s_j results from the union of s_k and s_l with $k, l < j$. If both s_k and s_l span blocks, then they are both found in S^b, and their images s_β and s_γ in $\psi(S^b)$ together form s_α. Suppose that s_k lies within one block. If that block is not the rightmost, then $\psi(s_k)$ is either the empty interval or a singleton. So $s_\alpha = \psi(s_k) \cup \psi(s_l)$ satisfies the definition of a scheme. If s_k lies inside the rightmost block, then it is possible that $\psi(s_k)$ is larger than a singleton, it could be that $\psi(s_k) = [z_{b-2}, x_{b-1}]$. However, then $s_\alpha = s_0 \cup \psi(s_l)$, and again the definition of a scheme is satisfied. Arguing the other cases similarly shows $\psi(S^b)$ is a scheme. As in the previous lemma, the facts that S solves Q and that $\psi \circ \psi_i$ is the identity on Z combine to show that $\psi(S^b)$ solves B.

Let q be an interval in the task Q of the form $\psi_i(z)$, but $i \neq 0$. That is, q is in the image of B, but not of that portion of B placed in the partition B_0. Let $i(q)$ be the index of the leftmost q_i in q. Let $W(i)$ be the index of the first element in S^b which contains q_i but contains no q_j with $j < i$. Since S solves Q, $W(i)$ is defined. We consider the equation $s^b_{W(i)} = s' \cup s''$. At least one of s' and s'' contains q_i, let us say s'. By selection of $W(i)$, s' is itself contained in the block containing q_i. Because i is not divisible by a, the image of this set under ψ must be the empty set. (It cannot be that this set contains x_{n-1}.) Hence $\psi(s^b_{W(i)})$ either appeared before in $\psi(S^b)$ or, because $\psi(S^b)$ is a scheme, it is a singleton. In either case, we can remove this element from the sequence $\psi(S^b)$ and it still will be a scheme. After removal, it still will solve B since that set contains no singletons. □

We now derive a lower bound on the length of S. Because $\varphi(S^i)$ solves A, $r_i \geq t\,|\,A\,|/3$, $i = 0, \ldots, b-1$. Taking the indicated subsequence of $\psi(S^b)$, we have a scheme of size $r_b + 1 - |\,B\,| + |\,B_0\,|$ solving B. Therefore, that sum is bound below by $(t-1)\,|\,B\,|/3$. Recall that each interval in B_0 ends over one of $z_0, \ldots, z_{b-3}$, and conversely, each $z_0, \ldots, z_{b-3}$ has at most one interval in B_0 ending over it. Hence $|\,B_0\,| \leq b-2$. Since $|\,Q\,| = b\,|\,A\,| + |\,B\,|$,

$$\begin{aligned} r &= r_0 + \cdots + r_{b-1} + r_b \\ &\geq bt\,|\,A\,|/3 + (t-1)\,|\,B\,|/3 + |\,B\,| - |\,B_0\,| - 1 \\ &\geq bt\,|\,A\,|/3 + (t+2)\,|\,B\,|/3 - (b-2) - 1 \\ &= t\,|\,Q\,|/3 + (2/3)\,|\,B\,| - b + 1. \end{aligned}$$

Because $B = \mathcal{T}_b(t-1, a)$ we have $|\,B\,| \geq ab/2$. Since $a \geq 3$,

$$r \geq t\,|\,Q\,|/3 + ab/3 - b + 1 > t\,|\,Q\,|/3$$

We have completed the verification of the three properties of $\mathcal{T}_n(t,k)$ enunciated in the lemma, and the induction step is complete.

Armed with Lemma 3.2.1, we state and prove the lower bound theorem. First we define $\alpha(m,n)$, the inverse Ackermann function. We follow Tarjan [92] in defining $A(i,j)$ for all $i \geq 1$ and $j \geq 0$ as:

$$\begin{aligned} A(1,j) &= 2^j, && j \geq 0 \\ A(i,0) &= 2, && i > 1 \\ A(i,j) &= A(i-1, A(i,j-1)), && j > 0,\ i > 1. \end{aligned}$$

The first row grows exponentially. The second row has at index j a tower of 2's $j+1$ high. That is, $A(2,0) = 2$ and increases by $A(2,j+1) = 2^{A(2,j)}$. Each row is a primitive recursive function growing faster than the previous one. But the real interest is in the growth of the columns. Each column past the first one grows roughly as fast as every other column, and all grow faster than any row. No column is primitive recursive, it simply grows too fast. Tarjan defined a functional inverse of $A(i,j)$. For all $m \geq n$:

$$\alpha(m,n) = \min\{\, i \mid A(i, \lfloor m/n \rfloor) > \log n \,\}.$$

We need to compare the functions $A(i,j)$ and $R(i,j)$.

Lemma 3.3.3 *For all $i,j = 1,2,\ldots$, we have $R(i+1,j) > A(i,j)$.*

We omit the proof which is by double induction. Combined with the following lemma it shows that $A(i,j)$ and $R(i,j)$ are essentially the same function.

Lemma 3.3.4 *For all $i,j = 1,2,\ldots$, we have $A(i+2,j) > R(i,j)$.*

Proof: We begin by showing that $A(i+1,j+1) > R(i,j)+2$, with $i = 1,2\ldots$ or $j = 0,1,\ldots$. Direct calculation shows this for $i = 1$ and $j = 0$. The induction step assumes $A(i'+1,j'+1) > R(i',j')+2$ if $i' < i$ or $i' = i$ and $j' < j$, and concludes with the inequality for $A(i+1,j+1)$. The following series of inequalities:

$$\begin{aligned} A(i+1,j+1) &= A(i, A(i+1,j)) \\ &\geq A(i, R(i,j-1)+2) \\ &> A(i, R(i,j-1)+1)^2 \\ &\geq (R(i,j-1)+2)R(i-1, R(i,j-1)) \\ &> R(i,j)+2 \end{aligned}$$

is justified in the remainder of this paragraph. The first inequality is an application of the induction hypothesis; the second uses the little result: for $i \geq 2$, $A(i,j+1) > A(i,j)^2$. This can be proven by induction. The next inequality uses the induction hypothesis and the result, $A(i,j) \geq j+1$, this for any $i,j \geq 1$. An easy induction shows that $A(i+1,j) \geq A(i,j+1)$, for $j \geq 1$, therefore, $A(i+2,j) \geq A(i+1,j+1) > R(i,j)$, and the lemma is proven. □

Suppose we are given m and n with $m \geq n$. Set $k = \lfloor m/n \rfloor$ and let t be the least integer such that $R(t,k) > n$. We explain why $t \geq 2$. A task cannot repeat a query interval, so $m \leq n(n-1)/2$, and hence,

$$R(1,k) = 2k \leq 2\lfloor m/n \rfloor \leq n-1.$$

By Lemma 3.2.1, there exists a task $T = \mathcal{T}_{n'}(t-1,k)$ with:

1. $n' = R(t-1,k)$, and it follows by the definition of t that $n' < n$.
2. T has size $|T|$ between $kn'/2$ and kn'.
3. Any solution to T has length at least $(t-1)|T|/3$.

Place $\lfloor n/n' \rfloor$ copies of T side by side. Add extra variables and queries to correct to form of this resultant task, that is, it will be over n variables and have m queries. The size of any solution is bounded from below by:

$$\begin{aligned}(t-1)\lfloor n/n' \rfloor |T|/3 &\geq (t-1)\lfloor n/n' \rfloor \lfloor m/n \rfloor n'/6 \\ &\geq m(t-1)/24 \\ &\geq mt/48.\end{aligned}$$

The following chain of inequalities shows that $t+2 \geq \alpha(m,n)$,

$$A(t+2,k) > R(t,k) > n > \log n.$$

Because $t \geq 2$, it follows that $t \geq \alpha(m,n)/2$. Therefore the cost of solving T is at least $m\alpha(m,n)/96$.

Consider now m and n given, where $m < n$. The previous paragraphs shows the existence of a task T over m variables with m queries which takes time at least $(m\alpha(m,m))/96$ to solve. If all the variables $x_0, \ldots, x_{m-1}$ do not appear in T, eliminate the unused variables and renumber as $x_0, \ldots, x_{m'-1}$. Add variables $x_{m'}, \ldots, x_{n-1}$. Find an interval of the form $[i, m'-1]$ in T, as $x_{m'-1}$ is used in T such an interval exists, and replace it with the interval $[i, n-1]$. Any solution to the resulting task can be made to solve the original task T by a transformation which includes removing the at least $n-m$ computation steps referencing variables with indices inside the interval $[m, n-1]$. So, the time to solve this new task must

be,

$$\begin{aligned} n - m + m\alpha(m,m)/96 &\geq (n-m)/192 + m\alpha(m,m)/96 \\ &\geq n/192 + m(2\alpha(m,m) - 1)/192 \\ &\geq (n + m\alpha(m,m))/192. \end{aligned}$$

We have therefore established:

Theorem 3.3.1 *Let X be a set of n variables, $x_0, \ldots, x_{n-1}$, and w a weight function from X to a faithful semigroup. For any $m \geq n$, there exists a set T of m intervals in X*

$$T \subseteq \{\, [x_i, x_j] \mid 0 \leq i < j < n \,\},$$

such that the length of any scheme solving T is $\Omega(m\alpha(m,n))$. For $m < n$ there exists such a T requiring a scheme of length $\Omega(n + m\alpha(m,m))$.

By a result of Yao [107] (see also Alon and Schieber [3] and Chazelle [17]), there exists an algorithm whose performance matches the lower bound.

Chapter 4

Simplex range reporting on a pointer machine

We give a lower bound on the complexity of the following problem, known as *simplex range reporting*: Given a set P of n points in d-space, precompute a data structure capable of reporting all points of $P \cap q$ where q is an arbitrary simplex. The *counting* variant has also been studied, and near optimal solutions exist. If m storage is available, algorithms with query time about $n/m^{1/d}$ have been discovered and discussed in the Chapter 2. For example, Chazelle, Sharir and Welzl [30] have proposed a $O(n^{d+\varepsilon})$ space, $O(\,(\log n)^{d+1}\,)$ time algorithm, and Matoušek [64] has demonstrated an $O(n)$ space, a $O(\,n^{1-1/d}(\log n)^{O(1)})\,)$ time algorithm. These demonstrate the two ends of the space-time tradeoff spectrum. A near matching lower bound in the arithmetic model of computation, Chazelle [19], shows these solutions to be quasi-optimal.

Here, the optimality of known algorithms in the reporting case is considered. Because the report itself takes time, it might be possible to customize the search to the number of points reported, giving a smaller data structure with no visible loss of query performance. This concept, called *filtering search*, Chazelle [15], was reviewed in the Chapter 2. We present a lower bound asserting that any data structure on a *pointer machine* exhibiting query time in $O(n^{\delta}+r)$ will occupy $\Omega(n^{d(1-\delta)-\varepsilon})$ storage, any fixed $\varepsilon > 0$. Once again, this differs from the upper bounds of Chazelle, Sharir and Welzl [30] and Matoušek [64] only by a factor of n^{ε}, and both these algorithms can run on a pointer machine, so these results are nearly tight.

The proof generalizes graph-theoretic arguments used in Chazelle [20] to establish a lower bound for orthogonal range reporting on a pointer machine. The rest of the chapter is organized as follows. In Section 4.1 we establish a combinatorial lemma which relates the size of the data structure to the cardinality of a special family of queries. In Section 4.2 such a special family is constructed: first, by positioning queries appropriately, then by placing points so as to satisfy certain properties with respect to the queries. The combinatorial lemma is applied and the lower bound results.

4.1 The complexity of navigation on a pointer machine

In the *pointer machine model of computation*, Tarjan [93], each memory cell includes a single data field and two pointer fields. The pointer fields reference other memory cells. Extensions to the model to include more data and pointer fields can be easily simulated within this definition. The computer accesses its memory by following pointers. It may read and modify the fields of the memory cells it accesses. We may extend the model to allow for the creation of new memory cells upon request. We give a more precise description of the machine's capabilities using a formal notation.

A data structure consists of a finite set V of nodes $\{ v_0, v_1, \ldots v_N \}$ and a set E of directed edges $(v, w) \in V^2$ such that the set of neighbors of any v,

$$N(v) = \{ w \in V \mid (v, w) \in E \},$$

has no more than two elements. An algorithm has a working set of nodes $W \subseteq V$. Initially $W = \{ v_0 \}$. The algorithm interacts with the memory by applying any of the following instructions:

1. Pick any $v \in W$ and add $N(v)$ to W.
2. Pick any $v, w \in W$ and, if $| N(v) | < 2$, add (v, w) to E.
3. Pick any edge $(v, w) \in E$, with $v, w \in W$, and remove it from E.

Each node v is associated with a datum, denoted $f(v)$. The algorithm may read or write the value of any vertex in its working set.

Let $P = \{ p_1, \ldots, p_n \}$ be a set of n points in E^d. To each node v is attached an integer $f(v)$. If $f(v) = i$ is not zero, then node v is associated with point p_i. A *query* q is a simplex

in E^d, and the algorithm must report all points in $P \cap q$. When presented with q, the algorithm begins with $W = \{ v_0 \}$ and terminates with a working set $W(q)$ that is required to contain the answer, namely,

$$\{ i \mid p_i \in q \} \subseteq \{ f(v) \mid v \in W(q) \}.$$

So, a query-answering algorithm must explore at least one node labeled with i for every p_i appearing in q. The cost of answering q is then $| W(q) |$. Note this is an underestimate of the computational cost, for it charges only one unit for every cell investigated, no matter how many times the cell was read from or written into or how much intermediate computation was necessary to decide which cell to visit. The *size* of the data structure V is the number of nodes in the graph. The model is easily extended to memory cells with any bounded number of data and pointer fields. Since the proof argues only about a "snapshot" of the data structure, extending the model by allowing for the creation or deletion of nodes is also possible without invalidating the proof.

Given a set $P = \{ p_1, p_2, \ldots, p_n \}$ of n points in E^d, a data structure $G = (V, E)$ is termed (a, δ)-*effective* if for any query q, we have $| W(q) | \leq a(| P \cap q | + n^{\delta})$. A collection of queries $Q = \{ q_i \}$ is called (c, k, δ)-*favorable* if for all i, and for all $i_1 < \cdots < i_k$,

1. $| P \cap q_i | > n^{\varepsilon}$.

2. $| P \cap q_{i_1} \cap \cdots \cap q_{i_k} | < c$.

We want to show that if δ is small, an (a, δ)-effective data structure must be large. Using the following lemma, we can bound their size by exhibiting a (c, k, δ)-favorable set of queries.

Lemma 4.1.1 *For any fixed $a, \delta > 0$ and $c > 1$, if G is (a, δ)-effective and Q is (c, k, δ)-favorable, then*

$$| V | > | Q | n^{\delta} / (4(k-1)2^{8c^2(a+1)}),$$

for n large enough.

Proof: We exploit the fact that the data structure can quickly answer a large number of very different queries to show that the data structure is itself large. More precisely, we look at the *c-sets* of V,

$$V^{(c)} = \{ W \subseteq V \mid | W | = c \}.$$

Recall that a tree is *rooted* if its edges are directed and the root is the only node with no incoming edge. Given any subset $w \subseteq V$, the *diameter* of w in V is the minimum number of edges in any rooted tree which spans w and is a subgraph of G. It is ∞ if no such tree exists. We denote the diameter of w by $\Lambda_G(w)$. This definition applies to any directed graph, in particular to subgraphs of G. Below we shall need Λ_T, where T is a rooted tree and a subgraph of G.

The number of c-sets in G of diameter smaller than r is bounded by,

$$\begin{aligned} \left| \{ w \in V^{(c)} \mid \Lambda_G(w) < r \} \right| &\leq \left| \{ (z, w) \in V \times V^{(c)} \mid \forall v \in w,\ d(z, v) < r \} \right| \\ &\leq |V| \binom{2^{r+1} - 1}{c} \\ &\leq |V| 2^{(r+1)c}, \end{aligned}$$

because of the limitation on the outdegree of G. Suppose now that query q is presented to the algorithm. Fix a rooted tree $T' \subseteq G$ which contains exactly the vertices $W(q)$. Because the algorithm reaches all the nodes in $W(q)$, such a tree exists. We can select from $W(q)$ a subset W such that

1. $|W| = |P \cap q|$, and
2. for every $p_i \in P \cap q$ there is a $w \in W$ such that $f(w) = i$.

Let T be the Steiner minimal tree for W inside of T'. Note that $\Lambda_T(w) \geq \Lambda_G(w)$ for any $w \subseteq G$. The tree T is rooted, therefore Λ_T satisfies another inequality: for any $w, w' \subseteq T$, $\Lambda_T(w \cup w') \leq \Lambda_T(w) + \Lambda_T(w')$.

Embed the tree T in the plane and number the vertices of W in their natural order around the border of T. Then, $W = w_1, w_2, \ldots, w_s$, where $s = |P \cap q|$, and

$$\sum_{j=1}^{s-1} \Lambda_T(\{w_j, w_{j+1}\}) \leq 2|T|.$$

Consider the c-sets,

$$W_i = \{ w_i, \ldots, w_{c+i-1} \}, \quad i = 1, \ldots, s - c + 1.$$

By the inequality discussed in the previous paragraph,

$$\Lambda_T(W_i) \leq \sum_{j=i}^{c+i-2} \Lambda_T(\{w_j, w_{j+1}\}).$$

Summing over all i,

$$\sum_{i=1}^{s-c+1} \Lambda_T(W_i) \leq (c-1) \sum_{j=1}^{s-1} \Lambda_T(\{w_j, w_{j+1}\}) \leq 2(c-1)|T|.$$

Since, $|T| \leq |W(q)|$, if we assume that G is (a,δ)-effective and Q is (c,k,δ)-favorable (thus $|P \cap q| > n^\delta$):

$$\sum_{i=1}^{s-c+1} \Lambda_T(W_i) < 4a(c-1)|P \cap q|,$$

for large enough n. By Markov's inequality,

$$|\{i \mid \Lambda_T(W_i) \geq 8a(c-1)\}| \leq |P \cap q|/2,$$

and therefore,

$$|\{i \mid \Lambda_T(W_i) < 8a(c-1)\}| > |P \cap q|/2 - c + 1.$$

Because $\Lambda_T(W_i) > \Lambda_G(W_i)$, this is also a lower bound on the number of c-sets with small diameter in G.

This argument is valid for any q in Q. Since $|P \cap q_{i_1} \cap \cdots \cap q_{i_k}| < c$, for appropriate indices $i_1 < \cdots < i_k$, a small c-set will be counted at most $k-1$ times. Thus,

$$\left|\{w \in V^{(c)} \mid \Lambda_G(w) < 8a(c-1)\}\right| > |Q||P \cap q|/(4(k-1)) > |Q|n^\delta/(4(k-1))$$

for large enough n.

In view of the upper bound given at the beginning of this proof, the result follows easily. □

4.2 A lower bound for simplex range reporting

According to the discussion of the previous section, any algorithm for solving simplex range reporting in time $O(n^\delta + r)$ can be modeled as an (a,δ)-effective data structure, for suitable a. The lower bound follows, therefore, from the construction of an set P of n points and a (c,k,δ)-favorable query set Q satisfying,

1. $|Q| = \Omega(n^{d(1-\delta)-\delta-\varepsilon})$, any fixed $\varepsilon > 0$.
2. $|P \cap q| > n^\delta$ for all $q \in Q$.
3. For each k distinct members $q_1, \ldots, q_k$ of Q, $|P \cap q_1 \cap \cdots \cap q_k| < c$.

Let $q \in E^d$ be any non-zero vector in Euclidean d-space. The hyperplane $H_q \subset E^d$ is defined by

$$H_q = \{ x \in E^d \,|\, \langle x, q \rangle - |q|^2 = 0 \}.$$

For any real $\mu > 0$, the *slab* $H_{q,\mu}$ is the set of all points within distance μ from H_q,

$$H_{q,\mu} = \{ x \in E^d \,||\, \langle x, q \rangle - |q|^2 | < \mu |q| \}.$$

The point q is the *defining point* of the slab $H_{q,\mu}$. Although our final result is stated for a collection of simplices, the query set we will construct is a collection of slabs. Once a favorable query set has been constructed, with slabs for queries, we can replace the slabs with simplices. We need that the points contained in the simplex are exactly those contained in the slab it replaces. This is easy to arrange by using very flat simplices.

We note a shift in our notation. The set Q will be a set of points. The collection of queries will actually be

$$\{ H_{q,\mu} \,|\, q \in Q \}.$$

The bijective correspondence between slabs and their defining points will render any ambiguity harmless.

Let $C_d = [0,1]^d$ be the unit d-cube in E^d. We construct a favorable query set in two steps. First we position the slabs so that their arrangement has certain geometric properties. Their intersection with C_d must be large, but their k-wise intersection with each other must be small. Next, n points are thrown at random into C_d and we verify that with high probability the slabs are favorable for this point set.

Further on we shall demonstrate that a sufficient condition for any k the slabs to intersect in a small volume is that any k of the defining points have a large convex hull. With this goal in mind we digress slightly in order to introduce a necessary result.

In about 1950, Heilbronn posed the following problem [71]: What is the largest area, over all point-sets $P = \{ p_1, \ldots, p_n \} \subset C_2$, of the smallest triangle with vertices in P? Heilbronn conjectured that this area is $O(1/n^2)$. However, Komlós, Szemerédi and Pintz [58] have shown the existence of point sets with small triangles of size $\Omega(\log n/n^2)$ and on the other hand they [57] have shown that n points in C_2 always form triangles of area less than $1/n^{8/7-\gamma}$ for any $\gamma > 0$. We shall require that the convex hull of k points in d dimensions contain volume $\Omega(1/n)$. This can be true if $k \geq \log n$. We recall the following result.

Theorem 4.2.1 (Chazelle[19]) *For any $d > 1$ there exists a constant $c > 0$ such that a random set of n points in C_d has, with probability greater than $1 - 1/n$, the property that the convex hull of any $k \geq \log n$ of these points has volume greater than ck/n.*

Hence a random point set is likely to be "good" for the construction of a favorable query set.

Having fixed a real $\gamma > 0$, let Q_0 be a random set of $1/\gamma^{d-1}$ points uniformly distributed in C_{d-1}. Theorem 4.2.1 assures that any $k \geq \log n$ points will enclose a large volume. Embed Q_0 in C_d via the map:

$$(x_1, \ldots, x_{d-1}) \mapsto \frac{1}{2}(x_1 + 1, \ldots, x_{d-1} + 1, 1).$$

Fixing a real $\mu > 0$, each image point gives rise to $\Theta(1/\mu)$ new points by the maps,

$$x \mapsto 2\mu z x,$$

where z ranges over all integers such that $1/2 \leq 2\mu z \leq 3/4$. To be precise, $z \in [i, j]$ with $i = \lceil 1/(4\mu) \rceil$ and $j = \lfloor 3/(8\mu) \rfloor$. Let Q be the image of Q_0 under the composition of these two maps. That is, the image of Q_0 under the map:

$$\begin{array}{rccc} \phi(x,z): & C_{d-1} \times [i,j] & \longrightarrow & C_d \\ & (x_1, \ldots, x_{d-1}, z) & \mapsto & \mu z(x_1 + 1, \ldots, x_{d-1} + 1, 2), \end{array}$$

see Figure 1.

Under the assumptions that μ and γ go to zero with increasing n, we show that:

1. Q is a set of size $\Theta(\, 1/(\mu\gamma^{d-1})\,)$.
2. For all $q \in Q$ the slabs $H_{q,\mu}$ have an intersection with C_d of volume $\Theta(\,\mu\,)$.
3. Any $k \geq \log n$ of these slabs have an intersection of volume $O(\,\mu^d(\log n/\gamma)^{d-1}\,)$ — and therefore the part of this volume which is inside C_d enjoys the same upper bound.

The first claim is trivial. The second follows from the fact that each coordinate of any $q \in Q$ is in the interval $[1/4, 3/4]$. So a ball of radius $1/4 - \mu$ and center q intersects H_q in a hyperdisk D which lies entirely inside C_d. The cylinder of height 2μ and cross section D at its midpoint is inside C_d. Here we assume, by increasing n if necessary, that $\mu \ll 1/4$. This gives the lower bound on the volume of $H_{q,\mu} \cap C_d$. The upper bound follows from placing

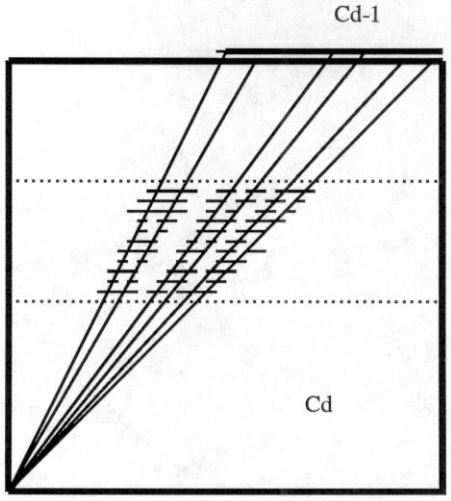

Figure 1: Building the query set.

a sufficiently large ball around q, say of radius $\sqrt{d}$, so as to contain the piece of $H_{q,\mu}$ which is also in C_d.

The third claim is substantiated as follows. Let $H_{q_1,\mu}, \ldots, H_{q_k,\mu}$ be the $k \geq \log n$ slabs, where the q_i are all distinct. If q_i and q_j are colinear with the origin, the intersection is empty. If they are not colinear, let $p_1, \ldots, p_k$ be the points in C_{d-1} which gave rise to $q_1, \ldots, q_k$. The convex hull of the p_i has volume at least $c_1 k \gamma^{d-1}$ for the appropriate constant c_1 given in Theorem 4.2.1. Triangulate the convex hull using $O(k^d)$ simplices and choose one among the simplices of largest area. After renumbering, the vertices of this simplex are $p_1, \ldots, p_d$ and it has area at least $c_2(\gamma/k)^{d-1}$. We conclude that $|\det(q_1, \ldots, q_d)| \geq c_3(\gamma/k)^{d-1}$, where the q_i have been renumbered according to the same pattern as the p_i and c_2 and c_3 are constants depending only on the dimension. The details of the argument are as follows. The first step in taking p_i to q_i involves compressing C_{d-1} by a constant factor then pasting it to the upper face of C_d. This affects the $d-1$-dimensional volume enclosed by the d points only by a constant factor. The cone on these d points with respect to the origin gives a simplex of volume equal to the area of the base, and this is a factor of $d!$ from the determinate of the matrix formed from the d points. The second step in taking p_i to q_i involves scaling each point by some bounded α_i, changing the determinate by only the product of these α_i.

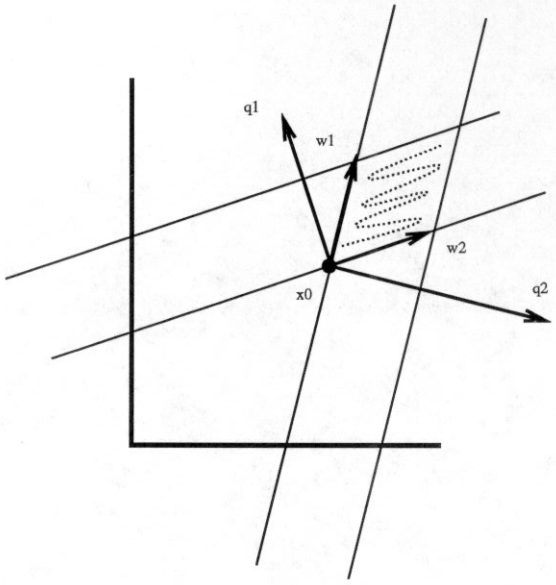

Figure 2: Intersection parallelotope.

Lemma 4.2.1 *For any $k \geq \log n$, every set $q_1, \ldots, q_k \subset Q$ contains a subset $q_{i_1}, \ldots, q_{i_d}$ such that,*

$$Vol(\, H_{q_1,\mu} \cap \cdots \cap H_{q_k,\mu}\,) \leq Vol(\, H_{q_{i_1},\mu} \cap \cdots \cap H_{q_{i_d},\mu}\,) = O(\, \mu^d (\log n/\gamma)^{d-1}\,).$$

Proof: The first inequality is trivial. In general, let $q_1, \ldots, q_d$ be linearly independent vectors. The polytope $H_{q_1,\mu} \cap \cdots \cap H_{q_d,\mu}$ is a translate of the parallelotope defined by d vectors w_j where,

$$\langle\, w_j, q_i \,\rangle = \begin{cases} 2\mu\,|\,q_i\,| & \text{if } i = j \\ 0 & \text{otherwise.} \end{cases}$$

To be more precise,

$$H_{q_1,\mu} \cap \cdots \cap H_{q_d,\mu} = \{\, \textstyle\sum_{i=1}^{d} \alpha_i w_i \,|\, 0 \leq \alpha_i \leq 1,\ i = 1, \ldots, d\,\} + x_o,$$

where x_o is the unique point of E^d satisfying,

$$\langle\, x_o, q_i \,\rangle - |\,q_i\,|^2 = -\mu\,|\,q_i\,|$$

for all $i = 1, \ldots, d$, see Figure 2. Denote by $[w]$ the matrix $(w_1, \ldots, w_d)$, by $[q]$ the matrix $(q_1, \ldots, q_d)$, and by Λ the diagonal matrix with $\Lambda_{ii} = |\,q_i\,|$. Note that $\det[w]$ is the volume

of the parallelotope. From $[w]^T[q] = (2\mu)^d \Lambda$ we have

$$\det[w] \det[q] = (2\mu)^d \, | \, q_1 \, | \cdots | \, q_d \, | \, .$$

Recall that from the set $q_1, \ldots, q_k \subset Q$ we can select d vectors such that $|\det[q]| \geq c_3(\gamma/k)^{d-1}$, for some constant c_3 depending only on the dimension, and $\sqrt{d}/2 \leq |\, q_i \,| \leq 3\sqrt{d}/4$. This gives the bound. □

The proof of the lower bound concludes with a probabilistic analysis of the interaction of n points chosen randomly in the unit cube C_d with the query set Q. For any real $0 < \delta < (d-1)/d$, and any $\varepsilon > 0$ set,

$$\mu = \frac{1}{\tau n^{1-\delta}}, \quad \gamma = \frac{1}{n^{1-\delta d/(d-1)-\varepsilon/(d-1)}},$$

where τ depends only on d and whose value will be fixed presently. Note that both μ and γ tend to 0 as n tends to infinity. Let $k = \log n$ be as before. We introduce the quantity,

$$c = \lceil d^2/\varepsilon \rceil.$$

We claim that the collection of slabs $H = \{ H_{q,\mu} \,|\, q \in Q \}$ is, with overwhelming probability, (c, k, δ)-favorable for the point set P.

Lemma 4.2.2 *Let the n points $P = \{ p_1, \ldots, p_n \}$ be independently and uniformly distributed in the unit cube C_d. With probability approaching 1 as n goes to infinity, for all $q \in Q$, $|\, H_{q,\mu} \cap P \,| > n^\delta$.*

Proof: The points $p_i \in H_{q,\mu}$, for $i = 1, \ldots, n$, are independent Bernoulli random variables with common probability,

$$p = \mathrm{Vol}\,(H_{q,\mu} \cap C_d) > K\mu = K/(\tau n^{1-\delta}),$$

for an appropriate K which depends only on d. This bound is a consequence of our second claim on Q, and we can adjust τ so that $np > 2n^\delta$. The expected number of points in q is therefore $E(|\, H_{q,\mu} \cap P \,|) = np > 2n^\delta$. The Chernoff bound [32] [44] states that, for $X = \{ x_1, x_2, \ldots \}$ a Bernoulli random variable where $x_i = 1$ with probability p and $x_i = 0$ with probability $1 - p$,

$$\mathrm{Prob}\left(\sum_{i=1}^{n} x_i \leq (1-\kappa)np \right) \leq \left(\frac{e^{-\kappa}}{(1-\kappa)^{1-\kappa}} \right)^{np},$$

for $0 < \kappa < 1$. Therefore, the probability that $|H_{q,\mu} \cap P| \leq np/2$ is less than $(2/e)^{np/2}$. Taking the disjunction over all $q \in Q$,

$$\begin{aligned} \text{Prob}\,(\exists\, q \in Q \text{ s.t. } |H_{q,\mu} \cap P| \leq np/2) &\leq |Q|\,\text{Prob}\,(|H_{q,\mu} \cap P| \leq np/2) \\ &< 1/(\mu\gamma^{d-1})\,(2/e)^{np/2} \\ &< n^{d(1-\delta)-\delta-\varepsilon}(2/e)^{n^\delta}. \end{aligned}$$

It is not difficult to see that this probability goes to 0 as n goes to infinity. Therefore, with probability approaching 1, every $H_{q,\mu}$ has more than $np/2 > n^\delta$ points in it. □

Lemma 4.2.3 *Let P be a set of n random points chosen uniformly in the unit cube C_d. With probability approaching 1 as n approaches infinity, for all $q_1, \ldots, q_k \in Q$, distinct,*

$$|H_{q_1,\mu} \cap \cdots \cap H_{q_k,\mu} \cap P| < c.$$

Proof: The events $p_i \in H_{q_1,\mu} \cap \cdots \cap H_{q_k,\mu}$, for $i = 1, \ldots, n$, are independent Bernoulli random variables with common probability,

$$\begin{aligned} p &= \text{Vol}\,(H_{q_1} \cap \cdots \cap H_{q_k} \cap C_d) < \text{Vol}\,(H_{q_{i_1},\mu} \cap \cdots \cap H_{q_{i_d},\mu} \cap C_d) \\ &< K\mu^d(\log n/\gamma)^{d-1} = K(\log n)^{d-1}/n^{1+\varepsilon}, \end{aligned}$$

for an appropriate constant K. Recall from Lemma 4.2.1 that this bound is a consequence of finding d from among the k slabs whose intersection is small. We formalize this choice in a function w from $Q^{(k)}$ to $Q^{(d)}$: for Υ a set of k slabs from Q, $w(\Upsilon) \subseteq \Upsilon$ is a set of d slabs whose existence and intersection volume is guaranteed by Lemma 4.2.1. We again refer to the Chernoff bound: for any positive real κ,

$$\text{Prob}\left(\sum_{i=1}^{n} x_i \geq (1+\kappa)np\right) \leq \left(\frac{e^\kappa}{(1+\kappa)^{1+\kappa}}\right)^{np},$$

thus if $np < 1$ then for any integer $c \geq 1$,

$$\text{Prob}\left(\sum_{i=1}^{n} x_i \geq c\right) \leq \left(\frac{enp}{c}\right)^c.$$

The expected number of points in $H_{q_{i_1},\mu} \cap \cdots \cap H_{q_{i_d},\mu}$ is less than 1 for n sufficiently large, hence,

$$\text{Prob}\left(\left|H_{q_{i_1},\mu} \cap \cdots \cap H_{q_{i_d},\mu} \cap P\right| \geq c\right) \leq \left(\frac{K(\log n)^{d-1}}{cn^\varepsilon}\right)^c.$$

Noting

$$\begin{aligned}&\text{Prob}\left(\exists q_1,\ldots,q_k \in Q^{(k)} \text{ s.t. } |H_{q_1,\mu}\cap\cdots\cap H_{q_k,\mu}\cap P| \geq c\right)\\ &\quad\leq \text{Prob}\left(\exists q_1,\ldots,q_d \in w(Q^{(k)}) \text{ s.t. } |H_{q_1,\mu}\cap\cdots\cap H_{q_d,\mu}\cap P| \geq c\right)\end{aligned}$$

we can bound the disjunction over all k-sets of queries in Q by

$$\begin{aligned}\text{Prob}\left(\exists q_1,\ldots,q_k \in Q^{(k)} \text{ s.t. } |H_{q_1,\mu}\cap\cdots\cap H_{q_k,\mu}\cap P| \geq c\right) &\leq \binom{|Q|}{d}\left(\frac{K(\log n)^{d-1}}{cn^{\varepsilon}}\right)^c\\ &\leq n^r(K/c)^c\end{aligned}$$

where

$$r = d(d(1-\delta)-\delta-\varepsilon)+c((d-1)(\log\log n/\log n)-\varepsilon).$$

By choice of c, r is negative, hence this probability vanishes as n approaches infinity. That is, with high probability, for all k distinct slabs, $|H_{q_1,\mu}\cap\cdots\cap H_{q_k,\mu}\cap P| < c$. □

What has been shown is the existence of a collection H of of $\Theta(n^{d(1-\delta)-\delta-\varepsilon})$ slabs and a set of n points P such that H is $(\lceil d^2/\varepsilon\rceil, \log n, \delta)$-favorable with respect to P. We can now apply Lemma 4.1.1 to give,

Theorem 4.2.2 *Simplex reporting on a pointer machine in E^d in time $O(n^{\delta}+r)$, where r is the number of points reported and $0<\delta\leq 1$, requires space $\Omega(n^{d(1-\delta)-\varepsilon})$ for any fixed $\varepsilon>0$.*

Proof: A pointer machine algorithm reporting points inside an arbitrary query simplex in time $O(n^{\delta}+r)$ would give an (a,δ)-effective data structure with $|V|$ nodes, for an appropriate a. If $\delta \geq (d-1)/d$ then the space bound is trivial, so assume otherwise. The randomized construction given above yields a (c,k,δ)-favorable query set H of size $\Theta(n^{d(1-\delta)-\delta-\varepsilon})$, where ε is any positive real, $c=\lceil d^2/\varepsilon\rceil$ and $k=\log n$. Lemma 4.1.1 then applies:

$$|V| \geq |H|\frac{n^{\delta}}{4(k-1)2^{8c^2(a+1)}} \geq K\frac{n^{d(1-\delta)-\varepsilon}}{256^{d^4(a+1)/\varepsilon^2}\log n},$$

for an appropriate K depending only on the dimension d and assuming n sufficiently large. □

Chapter 5

Conclusions

In this thesis we studied algorithms and lower bounds for geometric range searching, particularly for the problems of reporting and counting points inside of axis-parallel rectangles and simplices in E^d. The literature was carefully reviewed and two new results were given. Some open problems and new research directions are suggested in these closing paragraphs.

We would like to have a lower bound for partial-sums in multi-dimensional arrays which generalizes the one found in Chapter 3. That is, given an d-dimensional array and a collection of m hyper-rectangles, form the sum over all entries inside each rectangle. Chazelle and the author [28] have given $O(\,\alpha(m,n)^d\,)$ as an upper bound but improving the lower bound beyond $\Omega(\,\alpha(m,n)\,)$ has not been possible.

With regards to the lower bound of Chapter 4 for simplex range reporting, our bound implies that if the search time is to be in $O(\,(\log n)^b + r\,)$, for b arbitrarily large and r the number of points reported, then the space will be in $\Omega(\,n^{d-\varepsilon}\,)$ for all fixed $\varepsilon > 0$. In practical terms, this means logarithmic query time is a hopeless dream! Even a modest $O(\sqrt{n}+r)$ time algorithm in 10-dimensional space would require n^5 storage! On the theoretical side, however, we believe that the bound could be improved to $\Omega(\,n^d/\mathrm{polylog}(n)\,)$, and leave this as an open problem. Also, closely matching upper bounds are sought.

On the subject of upper bounds, for orthogonal range reporting on a pointer machine, an algorithm using $O(\,n(\log n/\log\log n)^{d-1}\,)$ space with query time $O(\,(\log n)^{d-1}+r\,)$ remains an open problem.

A lower bound on a pointer machine for halfspace range reporting would be very interesting. The methods used in the simplex reporting case should be applicable, but as yet

we have not succeeded at this. Our conjecture is a bound of $\Omega(n^{\lfloor d/2 \rfloor - \varepsilon})$ for polylog-time queries.

Our results hold for weights in a semigroup. For the static problem in higher dimensions with weights in a group very little is known. This is an important problem with many applications. Also, our results are for general semigroups. Perhaps the lower bound does not hold for some important special class of semigroups, for instance, *regular semigroups* where every element a has a "pseudo-inverse", namely, an element $a^\dagger$ which satisfies $aa^\dagger a = a$. The multiplicative semigroup of matrices is a regular semigroup.

The semigroup $(\mathbf{Z}, \min)$ has a special structure, in that $\min(a, b)$ is either a or b. This makes possible the following $O(n)$ space, $O(1)$ time algorithm for one-dimensional range counting. Build a tree $\mathcal{T}$ over the weighted points $p_1, \ldots, p_n$ by placing the the point p_i with minimum weight at the root and recursing in the left-son with points $p_1, \ldots, p_{i-1}$ and in the right-son with points $p_{i+1}, \ldots, p_n$. At the bottom of $\mathcal{T}$, add leaves and put them in one-to-one correspondence with the intervals between consecutive points. If $[a, b]$ has a in the interval corresponding to leaf l_a, and b to that of l_b, then the sum $\min_{p \in [a,b]} w(p)$ is the value stored in the least common ancestor node of l_a and l_b in $\mathcal{T}$. Least common ancestors are computable in $O(1)$ on a RAM by an algorithm of Harel and Tarjan [50]. This does not indicate a defect in our lower bound: the data structure is not universal, the tree is not the same shape for all weight assignments. However, we did want to signal how crucially the theory depends on its suppositions.

Bibliography

[1] P. Agarwal, M. Sharir, and P. Shor. Sharp upper and lower bounds on the length of general Davenport-Schinzel sequences. *Journal of Combinatorial Theory, Series A*, 52(2), 1989.

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

[3] N. Alon and B. Schieber. Optimal preprocessing for answering on-line product queries. Technical Report 71/87, The Moise and Frida Eskenasy Institute of Computer Science, Tel Aviv University, 1987.

[4] P. Assouad. Densité et dimension. *Ann. Inst. Fourier, Grenoble*, 33:233–282, 1983.

[5] D. Avis. On the partitionability of point sets in space. In *Proceedings of the Symposium on Computational Geometry*, pages 116–120, 1985.

[6] M. Avriel and D. J. Wilde. Optimality proof for the symmetric Fibonacci search technique. *Fibonacci Quarterly*, 4:265–269, 1966.

[7] H. S. Baird. Fast algorithms for LSI artwork analysis. *Journal of Design Automation and Fault-Tolerant Computing*, 2:179–209, 1978.

[8] M. Ben-Or. Lower bounds for algebraic computation trees. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 80–86, 1983.

[9] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9):509–516, 1975.

[10] J. L. Bentley. Decomposable searching problems. *Information Processing Letters*, 8(5):244–251, 1979.

[11] J. L. Bentley. Multidimensional divide and conquer. *Communications of the ACM*, 23(4):214–229, 1980.

[12] J. L. Bentley and J. B. Saxe. Decomposable seaching problems I: Static to dynamic transformations. *Journal of Algorithms*, 1(4):301–358, 1980.

[13] M. Bern. Hidden surface removal for rectangles. *Journal of Computer and System Sciences*, 40(1):49–69, 1990.

[14] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. Warmuth. Learnability and the Vapnik-Chervonenkis dimension. *Journal of the ACM*, 36(4):929–965, 1989.

[15] B. Chazelle. Filtering search: A new approach to query-answering. *SIAM Journal on Computing*, 15(3):703–724, 1986.

[16] B. Chazelle. Reporting and counting segment intersections. *Journal of Computer and System Sciences*, 32(2):156–182, 1986.

[17] B. Chazelle. Computing on a free tree via complexity-preserving mappings. *Algorithmica*, 2(3):337–361, 1987.

[18] B. Chazelle. A functional approach to data structures and its use in multidimensional searching. *SIAM Journal on Computing*, 17(3):427–462, 1988.

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

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

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

[22] B. Chazelle and H. Edelsbrunner. Optimal solutions for a class of point retrieval problems. *Journal of Symbolic Computation*, 1(1):47–56, 1985.

[23] B. Chazelle and H. Edelsbrunner. Linear space data structures for two types of range search. *Discrete & Computational Geometry*, 2(2):113–126, 1987.

[24] B. Chazelle and J. Friedman. A determinstic view of random sampling and its use in geometry. Technical Report CS–TR–181–88, Princeton University Department of Computer Science, 1988.

[25] B. Chazelle and L. Guibas. Fractional cascading: II. Applications. *Algorithmica*, 1(2):163–191, 1986.

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

[27] B. Chazelle and F. P. Preparata. Halfspace range search: An algorithmic application of k-sets. *Discrete & Computational Geometry*, 1(1):83–93, 1986.

[28] B. Chazelle and B. Rosenberg. Computing partial sums in multidimensional arrays. In *The Fifth Annual Symposium on Computational Geometry*, pages 131–139, 1989.

[29] B. Chazelle and B. Rosenberg. The complexity of computing partial sums off-line. *International Journal of Computational Geometry and Applications*, 1(1):33–45, 1991.

[30] B. Chazelle, M. Sharir, and E. Welzl. Quasi-optimal upper bounds for simplex range searching and new zone theorems. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, pages 23–33, 1990.

[31] B. Chazelle and E. Welzl. Quasi-optimal range searching and VC-dimension. *Discrete & Computational Geometry*, 4(5):467–490, 1989.

[32] H. Chernoff. A measure of asymptotic efficiency for tests of a hypothesis based on the sum of observations. *The Annals of Mathematical Statistics*, 23:137–142, 1952.

[33] K. L. Clarkson. New appplications of random sampling in computational geometry. *Discrete & Computational Geometry*, 2(2):195–222, 1987.

[34] K. L. Clarkson. Applications of random sampling in compuational geometry II. In *Proceedings of the Fourth Annual Symposium on Computational Geometry*, pages 1–11, 1988.

[35] R. Cole. Partitioning points sets in 4-dimensions. Technical Report 142, New York Univeristy, 1984. Technical Report, Department of Computer Science.

[36] R. Cole and C. K. Yap. Geometric retrieval problems. *Information and Control*, 63(1–2):39–57, 1984.

[37] D. Dobkin and R. J. Lipton. Multidimensional searching problems. *SIAM Journal on Computing*, 5(2):181–186, 1976.

[38] D. Dobkin and R. J. Lipton. On the complexity of computations under varying set of primitives. *Journal of Computer and System Sciences*, 18(1):86–91, 1979.

[39] H. Edelsbrunner. *Algorithms in Combinatorial Geometry*, volume 10 of *EATCS Monographs on Theoretical Computer Science*. Springer-Verlag, 1987.

[40] H. Edelsbrunner, D. G. Kirkpatrick, and H. A. Maurer. Polygonal intersection searching. *Information Processing Letters*, 14(2):74–79, 1982.

[41] H. Edelsbrunner and E. Welzl. On the number of line separations of a finite set in the plane. *Journal of Combinatorial Theory Series A*, 38:15–29, 1985.

[42] H. Edelsbrunner and E. Welzl. Halfplanar range seach in linear space and $O(n^{0.695})$ query time. *Information Processing Letters*, 23(6):289–293, 1986.

[43] P. Erdös, L. Lovász, A. Simmons, and E. G. Straus. Dissection graphs of planar point sets. In J. N. Srivastava, editor, *Survey of Combinatorial Theory*, pages 139–149. North-Holland, 1973.

[44] P. Erdös and J. Spencer. *Probabilistic Methods in Combinatorics*. Academic Press, 1974.

[45] M. L. Fredman. A lower bound on the complexity of orthogonal range queries. *Journal of the ACM*, 28(4):696–705, 1981.

[46] M. L. Fredman. Lower bounds on some optimal data structures. *SIAM Journal on Computing*, 10(1):1–10, 1981.

[47] M. L. Fredman. The spanning bound as a measure of range query complexity. *Journal of Algorithms*, 1(1):77–87, 1981.

[48] H. Fuchs, Z. M. Kedem, and B. F. Naylor. On visible surface generation by *a priori* tree structures. *Computer Graphics*, 14(3):124–133, 1980.

[49] B. Grünbaum. *Convex Polytopes*, volume 14 of *Pure and Applied Mathematics*. Interscience Publishers, 1967.

[50] D. Harel and R. E. Tarjan. Fast algorithms for finding nearest common ancestors. *SIAM Journal on Computing*, 13(2):338–355, 1984.

[51] S. Hart and M. Sharir. Nonlinearity of Davenport-Schinzel sequences and of generalized path compression schemes. *Combinatorica*, 6(2):151–177, 1986.

[52] D. Haussler and E. Welzl. ε-nets and simplex range queries. *Discrete & Computational Geometry*, 2(3):237–256, 1987.

[53] J. Kiefer. Sequential minimax search for a maximum. *Proceedings of the AMS*, 4:502–506, 1953.

[54] D. E. Knuth. *The Art of Computer Programming*, volume 3. Addison-Wesley Publishing, 1973.

[55] A. N. Kolmogorov. On the notion of algorithm. *Uspehi Mat. Nauk.*, 8, 1953.

[56] A. N. Kolmogorov and V. A. Uspenskii. On the definition of an algorithm. *Amer. Math. Soc. Transl.*, 29, 1963.

[57] J. Komlós, E. Szemerédi, and J. Pintz. On Heilbronn's triangle problem. *Journal of the London Mathematical Society*, 24(2):385–396, 1981.

[58] J. Komlós, E. Szemerédi, and J. Pintz. A lower bound for Heilbronn's problem. *Journal of the London Mathematical Society*, 25(2):13–24, 1982.

[59] U. Lauther. Four-dimensional binary search trees as a means to speed up associative searches in the design verification of integrated circuits. *Journal of Design Automation and Fault-Tolerant Computing*, 2(3):241–247, 1978.

[60] D. T. Lee and C. K. Wong. Worst-case analysis for region and partial region searches in multidimensional binary search trees and balanced quad trees. *Acta Informatica*, 9(1):23–29, 1977.

[61] G. S. Lueker. A data structure for orthogonal range queries. In *Proceedings of the 19th Annual IEEE Symposium on Foundations of Computer Science*, pages 28–34, 1978.

[62] J. Matoušek. Construction of ε-nets. *Discrete & Computational Geometry*, 5(5):427–448, 1990.

[63] J. Matoušek. Cutting hyperplane arrangements. In *Proceedings of the Sixth Annual Symposium on Computational Geometry*, pages 1–9, 1990.

[64] J. Matoušek. Efficient partition trees. In *Proceedings of the Seventh Annual Symposium on Computational Geometry*, pages 1–9, 1991.

[65] E. M. McCreight. Priority search trees. *SIAM Journal on Computing*, 14(2):257–276, 1985.

[66] M. McKenna. Worst-case optimal hidden-surface removal. *ACM Transactions on Graphics*, 6(1):19–28, 1987.

[67] P. McMullen and G. C. Shephard. *Convex Polytopes and the Upper Bound Conjecture*, volume 3 of *London Mathematical Society Lecture Note Series*. Cambridge University Press, 1971.

[68] C. A. Mead and L. Conway. *Introduction to VLSI Systems*. Addison-Wesley, 1979.

[69] K. Mehlhorn. *Data Structures and Algorithms 1: Sorting and Searching*. Springer-Verlag, 1984.

[70] K. Mehlhorn. *Data Structures and Algorithms 3: Multidimensional Searching and Computational Geometry*. Springer-Verlag, 1984.

[71] W. O. J. Moser. Problems on extremal properties of a finite set of points. *Discrete Geometry and Convexity*, 440:52–64, 1985.

[72] K. Mulmuley. An efficient algorithm for hidden surface removal. *Computer Graphics*, 23(3):379–388, 1989.

[73] K. Mulmuley. Hidden surface removal with respect to a moving view point. In *Proceedings of the Twenty Third Annual ACM Symposium on Theory of Computing*, pages 512–522, 1991.

[74] I. Nievergelt and E. M. Reingold. Binary search trees of bounded balance. *SIAM Journal on Computing*, 2(1):33–43, 1973.

[75] L. T. Oliver and D. J. Wilde. Symmetric sequential minimax search for a maximum. *Fibonacci Quarterly*, 2:169–175, 1964.

[76] M. Overmars and M. Sharir. Output sensitive hidden surface removal algorithms. In *Proceedings of the Thirtieth Annual IEEE Symposium on Foundations of Computer Science*, pages 598–603, 1989.

[77] M. H. Overmars. The design of dynamic data structures. In *Lecture Notes in Computer Science*, volume 156, 1983.

[78] M. S. Paterson and F. F. Yao. Point retrieval for polygons. *Journal of Algorithms*, 7(3):441–447, 1986.

[79] M. S. Paterson and F. F. Yao. Binary partitions with applications to hidden-surface removal and solid modeling. In *Proceedings of the Fifth Annual Symposium on Computational Geometry*, pages 23–32, 1989.

[80] M. S. Paterson and F. F. Yao. Optimal binary space partitions for orthogonal objects. In *Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 100–106, 1990.

[81] F. Preparata and M. I. Shamos. *Computational Geometry: An Introduction.* Texts and Monographs in Computer Science. Springer-Verlag, 1985.

[82] M. O. Rabin. Proving simultaneous positivity of linear forms. *Journal of Computer and System Sciences*, 6(6):639–650, 1972.

[83] J. Reif and S. Sen. An efficient output-sensitive hidden-surface removal algorithm and its parallelization. In *Proceedings of the Fourth Annual Symposium on Computational Geometry*, pages 193–200, 1988.

[84] E. M. Reingold. On the optimality of some set algorithms. *Journal of the ACM*, 19(4):649–659, 1972.

[85] P. Samuel. *Projective Geometry.* Readings in Mathematics. Springer-Verlag, 1988.

[86] N. Sauer. On the density of families of sets. *Journal of Combinatorial Theory: Series A*, 13(1):145–147, 1972.

[87] A. Schönhage. Real-time simulation of multidimensional turing machines by storage modifications machines. Technical Report 37, Project MAC Technical Memorandum, MIT, 1973.

[88] M. I. Shamos. Geometry and statistics: Problems at the interface. In J. F. Traub, editor, *Symposium on New Directions and Recent Results in Algorithms and Complexity.* Academic Press, 1976.

[89] J. M. Steele and A. C. Yao. Lower bounds for algebraic decision trees. *Journal of Algorithms*, 3(1):1–8, 1982.

[90] J. Stolfi. Primitives for computational geometry. Technical Report 36, Digital Equipment Corportaion, 1989.

[91] I. E. Sutherland, R. F. Sproull, and R. A. Schumacker. A characterization of ten hidden-surface algorithms. *Computing Surveys*, 6(1):1–55, 1974.

[92] R. E. Tarjan. Efficiency of a good but not linear set union algorithm. *Journal of the ACM*, 22(2):215–225, April 1975.

[93] R. E. Tarjan. A class of algorithms which require nonlinear time to maintain disjoint sets. *Journal of Computer and System Sciences*, 18(2):110–127, 1979.

[94] R. E. Tarjan. *Data Structures and Network Algorithms*, volume 44 of *CBMS-NSF Regional Conference Series in Applied Mathematics.* Society for Industrial and Applied Mathematics, 1983.

[95] P. M. Vaidya. Space-time tradeoffs for orthogonal range queries. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 169–174, 1985.

[96] V. N. Vapnik and A. Ya. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theoretical Probability and its Applications*, 16(2):264–280, 1971.

[97] E. Welzl. More on k-sets of finite sets in the plane. *Discrete & Computational Geometry*, 1(1):95–100, 1986.

[98] E. Welzl. Partition trees for triangle counting and other range searching problems. In *Proceedings of the Fourth Annual Symposium on Computational Geometry*, pages 23–33, 1988.

[99] D. E. Willard. *Predicate-oriented Database Search Algorithms.* PhD thesis, Harvard University, 1978. TR-20-78.

[100] D. E. Willard. Polygon retrieval. *SIAM Journal on Computing*, 11(1):149–165, 1982.

[101] D. E. Willard. New data structures for orthogonal range queries. *SIAM Journal on Computing*, 14(1):232–253, 1985.

[102] D. E. Willard. Lower bounds for dynamic range query problems that permit subtraction. In *Proceedings of the 13th International Colloquium on Automata, Languages and Programming*, pages 444–453, 1986.

[103] D. E. Willard and G. S. Lueker. Adding range restriction capability to dynamic data structures. *Journal of the ACM*, 32(3):597–617, 1985.

[104] A. C. Yao. A lower bound for finding convex hulls. *Journal of the ACM*, 28(4):780–787, 1981.

[105] A. C. Yao. Should tables be sorted? *Journal of the ACM*, 28(3):615–628, 1981.

[106] A. C. Yao. Space-time tradeoff for answering range queries. In *Proceedings of the 14th Annual ACM Symposium on Theory of Computing*, pages 128–136, 1982.

[107] A. C. Yao. On the complexity of maintaining partial sums. *SIAM Journal on Computing*, 14(2):277–288, 1985.

[108] A. C. Yao and F. Yao. A general approach to geometric queries. In *Proceedings of the 17th Annual ACM Symposium on Theory of Computing*, pages 163–168, 1985.

[109] F. Yao. A 3-space partition and its applications. In *Proceedings of the 15th Annual ACM Symposium on Theory of Computing*, pages 258–263, 1983.